DR. ÚRY LÁSZLÓ

COMMODORE 128

II. KÖTET

Programozói kézikönyv



LSI ALKALMAZÁSTECHNIKAI
TANÁCSADÓ SZOLGÁLAT

DR. ÚRY LÁSZLÓ

COMMODORE 128

II. KÖTET

Programozói kézikönyv

LSI
ALKALMAZÁSTECHNIKAI TANÁCSADÓ SZOLGÁLAT
BUDAPEST, 1988

Tartalomjegyzék

Bevezetés

A Commodore 128 I. kötet a számítógép működésének alapjait tartalmazta, s elsősorban a BASIC V7.0 programnyelvet ismertette. A II. kötet a gép programozásához szükséges további ismereteket tartalmazza. A C-64-es üzemmódról csak elvétve esik szó, a könyv elsősorban a CP/M üzemmód működésével foglalkozik.

A C-128-as üzemmódról szóló rész tartalmazza a 80 oszlopos video chip részletes leírását, továbbá az új 1571/70-es lemezegységek használatát.

A C-128-as rész CP/M rendszerének fejlesztése még jelenleg is tart. A könyv az 1986 év végi állapotot ismerteti, az újabban vásárolt gépek operációs rendszere már eltérhet attól, elsősorban abban, hogy többet tud.

A Commodore 128-cal adott CP/M rendszerlemez önmagában semmit sem ér, hiszen egyetlen program-fejlesztő eszközt sem tartalmaz. A szóbajöhető programnyelvek közül végül is a Turbo Pascalt választottam ki, elsősorban azért, mert használata egyszerű, s a programnyelv majdnem ugyanilyen formában működik a 16 bites gépeken is. Úgy vélem, hogy a Commodore 128-as gépet CP/M-ben elsősorban Turbo Pascal-ban kell programozni. Ezért erről a programozási nyelvről, s főleg a C-128-as sajátosságairól részletesen szólunk a 13. és 14. fejezetekben.

A 15. fejezet két makro assembler fejlesztő rendszert ismertet. A használt Z80-as mnemonikok kiválasztása nem volt könnyű. A Commodore 128-hoz a Digital Research által kifejlesztett RMAC fejlesztő rendszert lehet 'hivatalosan' megvásárolni, ugyanakkor a Z80-as processzor miatt szükség lehet a ZSID-szerű mnemonikok használatára. Az RMAC és a hozzá tartozó egyéb rendszerprogramok segítségével pl. nem lehet vissza assemblálni a CP/M programokat, mert a BIOS tartalmaz olyan Z80-as utasításokat is, amelyek nincsenek benne az Intel 8080 processzor utasításaiban is. Az eltérő mnemonikok használatát sem könnyű megszokni. A részletek iránt érdeklődőknek javasoljuk **Hofmanné Boskovitz Éva: Z80 assembler** című, az LSI ATSZ gondozásában megjelent könyvét.

8. fejezet

A C-128-as operációs rendszer

8.1 A társprocesszorok működése

A Z80A jelű mikroprocesszor a Zilog Z80-as processzor 4MHz-es változata. A Z80 és a 8502-es processzorok társprocesszorként dolgoznak, ami azt jelenti, hogy ugyanazt a buszt használják. Ennek egyik következménye, hogy egyszerre csak egyikük dolgozhat. A C-128 hardverének kialakítása során a közös busz elérésének a biztosítása lehetett a legnehezebb. Ennek két oka van. Egyrészt a két processzor eltérő módon kezeli a buszt, másrészt a VICIIa video chip közvetlenül ír és olvas a memóriából. A megoldás 'végül' is egyszerű: a PLA gondoskodik arról, hogy abban az esetben ha a buszon érvénytelen cím vagy adat van (a 8502-es címzési eljárásából adódóan), a Z80A ne dolgozzon. Ez a Z80A sebességét a felére csökkenti. A megoldás sokkal megbízhatóbb, mint a C-64-hez kapható Z80-as cartridge esetén. A C-128 nem szokott 'lemerevedni'.

A számítógép bekapcsolásakor, vagy hardver reset (a RESET gomb megnyomása) esetén a vezérlés a Z80A processzorra kerül s az egy RST 00 utasítással megkezdi a működését. A Z80A választását a 8502 eltérő busz kezelése indokolja; ellenkező esetben (lásd a Commodore 64-et) a rendszer könnyen lemerevedhet. A Z80A bekapcsolásával egyidőben a hardver gondoskodik arról, hogy a processzor memória területén megjelenjen a 4K-s Z80 ROM. Ennek következtében a számítógép bekapcsolásakor a Z80 ROM 0-ás címétől kezd a rendszer futni. A minket érdeklő programrész listája (ZSID-del vissza assemblált kód!):

Z80-as kód:

```
0000  ld a,3e          ; 0-ás szelet és I/O terület
      ld (ff00),a      ; bekapcsolása
      jp 003b          ; ugrás az rst 00 további részére

003b  ld  bc,d02f      ; a VICIIa chip 47. regisztere
      ld  de,fffc      ; d-ben a billentyűzet lekérdezése
      out (c),d        ; letiltva
      inc bc           ; a VICIIa chip 48. regisztere
      out (c),e        ; 1MHz-es órajel
      ld  bc,d505      ; mód konfigurációs regiszter
      ld  a,b0         ; /EXROM és /GAME ellenőrzése
      out (c),a        ; a megfelelő bitek bemenetek
      in  a,(c)        ; a bitek olvasása
      cpl              ; invertálás
      and 30           ; a két bit kiszűrése
      jr  z,0059       ; ha egyik sem 0, akkor nincs
                       ;  a rendszerben C-64-es cartridge
      ld  a,f1         ; 8502 és a C-64 mód kijelölése és
      out (c),a        ;  bekapcsolása
      rst 00
```

```
0059  ld  bc,dc0f        ; a CIA#1 CRB kijelölése
      ld  a,08           ;  és az A és B időzítők beállítása
      out (c),a          ;  a B időzítő
      dec c              ;  CRA kijelölése
      out (c),a          ;  az A időzítő
      ld  c,03           ; a DDRB kijelölése
      xor a              ; akku=0
      out (c),a          ; a B kapu bitjei bemenetek lesznek
      dec c              ; a DDRA kijelölése
      dec a              ; akku=ff
      out (c),a          ; az A kapu bitjei kimenetek lesznek
      dec c              ; PORTA kijelölése
      dec c              ;
      ld  a,7f           ; a <C=>-ot tartalmazó oszlop
      out (c),a          ;  kiválasztása
      inc bc             ; a PORTB kijelölése és
      in  a,(c)          ;  az adatok beolvasása
      and 20             ;  a <C=>-nak megfelelő bit kiszűrése
      ld  bc,d505        ; mód konfigurációs regiszter kijelölése
      jr  z,0054         ; ha a <C=> billentyű le volt nyomva
                         ;  áttérés C-64-es módba

007c  ld  hl,0fb4        ; az MMU 11 regiszterének beállítása:
      ld  bc,d50a        ;  másolás a 0fb4-től visszafelé
      ld  d,0b           ;  a d50a-ba (szintén visszafelé)
0084  ld  a,(hl)         ; egy byte átírása
      out (c),a
      dec hl             ; mutatók csökkentése
      dec c
      dec d
      jr  nz,0084
      ld  hl,0d1a        ; 0d1a-tól
      ld  de,1100        ; 1100-ra
      ld  bc,0008        ; 8 byte másolása
      ldir
      ld  hl,0ee5        ; 0ee5-től
      ld  de,ffd0        ; ffd0-ra
      ld  bc,001f        ; 31 byte másolása
00a0  ldir
00a2  ld  hl,1100        ; hl-be kerül a 1100 mutató
      ld  (fffa),hl      ; hl másolása fffa-ba
      ld  (fffc),hl      ;  fffc-be
      ld  (fffe),hl      ;  fffe-be és
00ae  ld  (ffdd),hl      ;  ffdd-be
00b1  jp  ffe0           ; a rst 00 végére ugrás
```

<u>8502-es kód:</u>

```
1100  lda #$00           ; összes ROM kijelölése és
      sta $ff00          ;  bekapcsolása
      jmp ($fffc)        ; ugrás a hardver RESET vektorra
```

```
ffd0  sei               ; megszakítások letiltása
      lda #$3e          ; a 0-ás RAM, I/O kijelölése és
      sta $ff00         ;  bekapcsolása
      lda #$b0          ; Z80-as processzor kijelölése és
      sta $d505         ;  bekapcsolása
      nop               ; kötelező üres utasítás
ffdc  jmp $3000         ; 8502-es program folytatása
      nop
```

Z80-as kód:

```
ffe0  di                ; megszakítások letiltása
      ld  a,3e          ; a 0-ás RAM, I/O kijelölése és
      ld  (ff00),a      ;  bekapcsolása
      ld  bc,d505       ; mód konfigurációs regiszter és
      ld  a,b1          ; a 8502-es processzor kijelölése majd
      out (c),a         ; a 8502-es bekapcsolása
      nop               ; kötelező üres utasítás
ffee  rst 38            ; a Z80-as kód folytatása
```

A RST 00 végrehajtása a 0-ás memóriaszelet és az I/O terület kijelölésével kezdődik. Ne felejtsük el, hogy a memória első 4K-s része azonban a Z80 ROM! Ezt követően a 003b-0052 rész ellenőrzi, hogy nincs-e C-64-es cartridge a gépben. Ha igen, a 0054-0056 utasítások bekapcsolják a C-64-es üzemmódot.
A 0059-007a részek megvizsgálják, hogy nincs-e lenyomva a <C=> billentyű. Ha igen, akkor újból a C-64-es mód bekapcsolására kerül a sor. Ellenkező esetben a 007c-00ae programrész a Z80-as ROM-ból két programrészt átmásol a 1100, illetve ffd0 címekre. Ezek közül az első az 8502-es kód, míg a második felerészben 8502-es, felerészben Z80-as kód. A két programrész a két processzor közti átkapcsolást biztosítja.

A 00a2-00ae közötti programrész a 8502 valamennyi hardver vektorába a 1100 mutató értékét tölti, majd a vezérlés az ffe0 címre kerül. Itt a rendszer bekapcsolja a 8502-es processzort, ami a 1100 címtől kezd el futni. Ide a Z80 egy rövid kódot másolt, ami bekapcsolja az összes ROM-ot és a hardver RESET vektorra ugrik.
A Z80 a NOP utasítás előtt állt meg. Ha a 8502-es ezek után bárhonnan visszaadja a vezérlést a Z80-nak csak úgy teheti, hogy a 0-ás lap ffee címén egy érvényes Z80-as utasítás van. A Z80 innen fogja a futását folytatni.

Az ffd0 programrész a fordított kapcsolatra való. Ha a jmp $ffd0 utasítással kapcsoljuk ki a 8502-est és indítjuk a Z80-at akkor a jp ffe0 utasítással visszakapcsolhatjuk a 8502-est. Ehhez azonban az ffdc címen egy 8502-es utasításnak kell lennie, mert ott fogja a futást a 8502-es folytatni.

Az alábbi BASIC program a 40 oszlopos képernyő bal felső sarkába ír egy a betűt. Ezt egyszerűen is megtehetnénk a POKE 1024,1 utasítással. Ehelyett bekapcsoljuk a Z80-as processzort, azzal iratjuk a memóriába az 'a' karakter képernyő kódját, majd visszakapcsoljuk a 8502-es processzort és visszatérünk a BASIC-be.

```
1000 bank 0
1010 poke dec("8000"),0
1020 cim=dec("ffee"): db=8
1030 gosub 1110
1040 cim=dec("ffdc"): db=2
1050 gosub 1110
1060 scnclr
1070 sys dec("ffd0")
1080 poke 1024,peek(dec("8000"))
1090 end
1100 :
1110 for i=cim to cim+db-1
1120 : read a$: poke i,dec(a$)
1130 next i
1130 return
1150 :
1160 rem z80 kod
1170 rem *******
1180 :
1190 data 3e,01   : rem ld a,1
1200 data 32,00,80: rem ld (8000),a
1210 data c3,e0,ff: rem jp ffe0
1220 :
1230 rem 8502 kod
1240 rem *******
1250 :
1260 data 58      : rem cli
1270 data 60      : rem rts
```

Az 1110-1140 sorokban található alprogram a READ utasítás segítségével beolvas db számú karaktert és a memória CIM-étől kezdve elhelyezi. Ezt az alprogramot kétszer hívtuk meg: a 8502-es, illetve a Z80-as kód átmásolásához. A program elején külön be kellett a BANK utasítással a megfelelő memóriaszeletet (0-ás szelet) állítani.

A Z80-as kód az A regiszterbe tölti az 1-et (az 'a' karakter képernyő kódját), majd ezt a 8000-es címre írja. Ezt követően a Z80-as visszaadja a vezérlést a 8502-es processzornak. Az a cli utasítással megengedi a megszakítások lekezelését és visszatér a BASIC-be.

Maga a BASIC program - a megfelelő memória tartalmak beállításán túl először törli a 8000-es memóriacímet, majd a SYS DEC("FFD0") utasítással átadja a vezérlést a Z80-as processzornak. A gépi kódú alprogramból való visszatérést követően a 8000-es memóriahely tartalmát átmásolja a képernyőmemória első címére. (Ez felel meg a bal felső karakternek.)

A Z80-as rutin nem tud közvetlenül a $0400-as címre írni. Ennek oka, hogy a 0-ás RAM és a Z80-as processzor használata esetén a $0000-$0fff címeken mindig a Z80-as ROM található. Ezért a Z80-as egy segédregiszteren keresztül írhat csak a C-128 BASIC interpreter által használt képernyő memória területére.

Ha a C-128-hoz vásárolható SID (Symbolic Instruction Debugger), vagy a C64 CP/M-jéhez adott DDT gépi kódú monitort használjuk, akkor nincs lehetőségünk arra, hogy a 0-ás lapon levő programrészeket olvashassuk. Ehhez ezeket a MOVE és XMOVE BIOS rutinok segítségével át kell másolnunk a 0. szeletről az 1. szeletre. Az alábbi egyszerű rutin a Z80-as ROM-ot másolja át az 1. szeletre a $5000 címtől kezdődően. (A $0000-ra nem tehetjük, mert akkor az a monitort felülírná. Megint a ZSID-del készítettük el a programot:

```
a4000
4000  ld   b,1          ; másolás az 1. szeletre
4002  ld   c,0          ; másolás a 0. szeletre
4004  call 0f8ab        ; XMOVE végrehajtása
4007  ld   de,0         ; az első másolandó karakter régi és
400a  ld   hl,5000      ;  új helye
400d  ld   bc,1000      ; átmásolandó karakterek száma, 1Kbyte
4010  call 0f8a0        ; másolás végrehajtása
4013  ret               ; vissza a monitorhoz
```

Abban az esetben ha a Digital Research SID monitorát használjuk, akkor a fenti asszembli rutint a következőképpen kell begépelni:

```
a4000
4000  mov  b,1          ; másolás az 1. szeletre
4002  mov  c,0          ; másolás a 0. szeletre
4004  call f8ab         ; XMOVE végrehajtása
4007  lxi  de,0         ; az első másolandó karakter régi és
400a  lxi  hl,5000      ;  új helye
400d  lxi  bc,1000      ; átmásolandó karakterek száma, 1Kbyte
4010  call f8a0         ; másolás végrehajtása
4013  ret               ; vissza a monitorhoz
```

A rutint a c4000 monitor paranccsal futtassuk le. (Ha a call monitor parancsot nem ismeri a rendszerünk, akkor ret helyett a rst 38 paranccsal fejezzük be a rutint, s g4000-rel indítsuk el!)

A rutin lefuttatása után a Z80 ROM a processzor memória területén az 5000 címtől megjelenik, s működését tanulmányozhatjuk. Mivel az eredeti ROM-ban van, ezért átírására természetesen nincs lehetőség.
Hasonló módszerrel a 0. szelet akármelyik részét átmásolhatjuk az 1. szeletre, s ott módosíthatjuk is.

8.2 A C-128-as üzemmód beállítása

A C-128-as üzemmód működése nagyon sok hasonlóságot mutat a Commodore 64 működésével. Az eltérések a következőkből adódnak:

- a képernyő szerkesztő működése;
- a memória szeletei közti lapozás;
- az eltérő inicializálás;
- a társprocesszorok használata;
- új KERNAL rutinok.

A képernyő szerkesztő eltérő működését csak jelezzük. Az osztott képernyő használata esetén a C-128 használja a raszter szerinti megszakítást is. Ez okozza a képernyő szerkesztő eltérő működését. Ebben az esetben a megszakítások sokkal gyakoribbak és nagyon pontosan kell lekezelni ezeket. További eltérés az ablak technika alkalmazása. Ennek következtében egyes KERNAL paraméterek az ablak bal felső sarkára, s nem a fizikai képernyő bal felső sarkára vonatkoznak.

A memória szeletek közti lapozás lelassítja a gép működését. A 2MHz-es órajel éppen hogy csak korrigálni tudja ezt a lassulást a C-64-es módhoz képest. A számításigényes vagy sok adatot mozgató programok gyorsabbak C-64-es üzemmódban, mint C-128 módban! A memória szeletek használatáról részletesen szóltunk a 7.3 fejezetben. A továbbiakban a C-128-as eltérő inicializálását ismertetjük.

Valahányszor a 8502-es processzor RESET lábán egy lefutó él megjelenik, a processzor egy indirekt vezérlésátadást hajt végre a $FFFC-ben tárolt vektorra. Ez mindannyiszor megtörténik, valahányszor bekapcsoljuk a számítógépet, vagy megnyomjuk a gép oldalán levő RESET gombot.

A 8502-es chip működését azonban - rögtön a RESET lábon megjelenő lefutó él megjelenésekor - a PLA felfüggeszti és a vezérlést a Z80A processzornak adja át. Mint az előző részben leírtuk, a Z80A ellenőrzi, hogy nincs-e a rendszerben C-64-es cartridge, vagy nincs-e megnyomva a <C=> billentyű. Ha igen, akkor a Z80A **közvetlenül** bekapcsolja a C-64-es módot. Ellenkező esetben a Z80A beállítja az MMU regisztereit, s átadja a vezérlést a 8502-esnek, ami ezután tudja csak a $FFFC vektorra való ugrást végrehajtani.
Amikor a 8502-es processzor elindul, akkor a START KERNAL rutin minden esetben végrehajtja a következő funkciókat:

- bekapcsolja az összes ROM-ot és a 0-ás RAM-ot;
- letiltja a megszakításokat;
- törli a processzor verem-mutatóját;
- törli a decimális módot;
- inicializálja az MMU-t;
- lemásolja a ROM-ból a RAM rutinokat;
- ellenőrzi és beállítja a SYSTEM vektort;
- ellenőrzi a rendszerbeli ROM-okat.

Az utolsó két lépés kivételével a KERNAL mindig pontosan ugyanazt végzi el. Azok eredménye azonban azon múlik, hogy a SYSTEM vektor (lásd később) mit tartalmaz, illetve, hogy milyen ROM cartridge-okat használunk. A SYSTEM rutin lehetőséget biztosít arra, hogy a KERNAL felismerje, hogy egy 'hideg' vagy 'meleg' újraindítás történt. Az utóbbi feltétele, hogy a $FFF5-$FFF7 címeken a CBM karaktersorozatot találja a rutin. Ebben az esetben a $FFF8-$FFF9 címeken levő vektort átmásolja a $02-es címre és végrehajt egy JMP($0002) utasítást. Ha nem találja a CBM sorozatot, akkor tovább folytatja a rendszer inicializálását. A SYSTEM KERNAL rutin a $FFF8 címen található.

A ROM-ok ellenőrzése ugyancsak a CBM karaktersorozat megkeresésével történik. A C-128 cartridge-ok első byte-jai a következőt kell, hogy tartalmazzák:

- $0000 hideg indítás;
- $0003 meleg indítás;
- $0006 azonosító kód;
- $0007-$0008 'CBM' karaktersorozat.

A byte-ok természetesen mindig a cartridge ROM kezdőcímétől értendőek. A lehetséges kezdőcímek: $8000, $C000. A ROM-ok keresése az alábbi sorrendben történik: külső alsó ROM ($8000, 16K vagy 32K), külső magas ROM ($C000, 16K), belső alsó ($8000, 16K vagy 32K) végül belső felső($C000,16K). A rendszer a cartridge-ok azonosítóit a $AC1-$AC4 táblázatba másolja. Az azonosítók egyben a cartridge-ok prioritását is megadják. Az alacsonyabb azonosítójú cartridge-nak nagyobb a prioritása. Az azonosító nem lehet 0, ugyanis a C-128 a $AC1-$AC4 táblázat nullától különböző értékeit tekinti jelen levő ROM-nak. Az 1 érték önindító ROM-nak felel meg, a rendszer automatikusan a cartridge hidegindítási pontjára adja a vezérlést egy JSR utasítással. Ha a cartridge végrehajt egy megfelelő RTS utasítást, akkor folytatódik a rendszer inicializálása. Különben a cartridge-nak kell teljes egészében gondoskodnia az inicializálásról. Az 1-től eltérő azonosítójú ROM-ok inicializálására később kerül sor.
A C-128 tovább folytatja az inicializálást:

- az IOINIT KERNAL rutin inicializálja a perifériákat;
- ellenőrzi a <STOP> és <C=> billentyűket;
- beállítja a 0-ás lapon levő indirekt ugrási címeket;
- a CINT KERNAL rutinnal inicializálja a képernyő szerkesztőt;
- BASIC kezdőértékek beállítása;
- belépés a BASIC várakozó ciklusába.

A leglényegesebb szerepe az IOINIT rutinnak van: ez állítja be mindkét CIA chip regisztereit, a VIC szeletét, a PAL kijelzést. Ugyancsak ez a rutin állítja be a SID, a VIC és a 8563 chip kezdőértékeit, letölti a karakterek alakját.
A rendszer indulását - a jelenlevő ROM-okon kívül - a <STOP> vagy a <C=> billentyű lenyomása befolyásolja. Az utóbbit már a Z80 ellenőrzi, így itteni ellenőrzése felesleges. Ha a <STOP> billentyűt lenyomott állapotban találja, akkor nem a BASIC-be, hanem a monitorba lép be a rendszer. Ennek következtében kimarad a BASIC rendszerváltozók inicializálása. Ez főleg akkor hasznos, ha egy gépi kódú programrész kipróbálása közben elszáll a rendszer, és minél kevesebb memóriarész törlésével akarjuk a rendszert a RESET gombbal újraindítani.

Az inicializálási eljárás legvégén kerül sor a BASIC rendszerváltozók inicializálására. Ez a programrész meghívja a PHONIX KERNAL rutint, ami valamennyi ROM-ot elindít, illetve a 8-as lemezegységet ellenőrzi: nincs-e önindítós (bootable) lemez benne.

A $A04 memóriacím ad felvilágosítást a rendszer inicializálásának előrehaladásáról. Ennek a regiszternek a megfelelő beállításával elérhetjük, hogy egyes részek kimaradjanak. A $A04-et a rendszer a következőképpen használja:

- $B7 8563 karakterek már átmásolva;
- $B6 CINT már végrehajtva;
- $B0 BASIC már inicializálva.

Ha az IOINIT-et meghívja a rendszer és $A04 tartalma $B7, akkor a rutin nem tölti át a 8563 karakterkészletét. Ha $B6 van beállítva, akkor a CINT nem írja felül a billentyűzet mátrix kódtábláit.

8.3 Memória térkép

Név	Cím hexa.	dec.	Leírás
D6510	0000	0	8510 processzor adat-irány regisztere
R6510	0001	1	8510 processzor adatregiszter
BANK	0002	2	a használt szelet száma
PC$HI	0003	3	a SYS címének felső byte-ja
PC$LO	0004	4	a SYS címének alsó byte-ja
S$REG	0005	5	a processzor ST byte-jának ideiglenes tárolása
A$REG	0006	6	a processzor A regiszterének ideiglenes tárolása
X$REG	0007	7	a processzor X regiszterének ideiglenes tárolása
Y$REG	0008	8	a processzor Y regiszterének ideiglenes tárolása
STKPTR	0009	9	a veremmutató (SP) tárolása
ENDCHR	000A	10	jelző: keresés a sztring végéig
TRMPOS	000B	11	az utolsó TAB-tól az oszlopok száma
VERCK	000C	12	jelző: 0=LOAD, 1=VERIFY
COUNT	000D	13	input puffer mutató, a tömb dimenziójának száma
DIMFLG	000E	14	Tömb default dimenziója (=10)
VALTYP	000F	15	Adattípus: $FF = sztring, $00 = numerikus
INTFLG	0010	16	Adattípus: $00 = valós, $80 = egész
GARBFL	0011	17	szemétgyűjtés jelzője, LIST, DATA használja
SUBFLG	0012	18	FN jelző
INPFLG	0013	19	$00 = INPUT, $40 = GET, $98 = READ
TANSGN	0014	20	TAN előjele
CHANNL	0015	21	aktiv I/O száma, input kérdés jelzője
LINNUM	0016-0017	22	sorszám jelzője
TEMPPT	001	24	ideiglenes sztring verem mutatója
LASTPT	0019-001A	25	utolsó sztring cím
TEMPST	001B-0023	27	verem a sztringleírók ideiglenes tárolására
INDEX1	0024-0025	36	mutató sztringműveletekhez
INDEX2	0026-0027	37	mutató sztringműveletekhez
RESHO	0028-002C	40	szorzás lebegőpontos eredménye
TXTTAB	002D-002E	45	mutató: BASIC szöveg eleje
VARTAB	002F-0030	47	mutató: BASIC változók kezdete
ARYTAB	0031-0032	49	mutató: BASIC tömbök kezdete
STREND	0033-0034	51	mutató: BASIC tömbök vége + 1
FRETOP	0035-0037	53	mutató: a sztring terület eleje
FRESPC	0037-0038	55	mutató sztringműveletekhez
MAXMEM1	0039-003A	57	mutató: a sztringterület vége az 1. szeleten
CURLIN	003B-003C	59	aktuális BASIC sorszám
TXTPTR	003D-003E	61	CHRGET mutatója
FORM	003F-0040	63	PRINT USING segédváltozója
DATLIN	0041-0042	65	aktuális DATA sorszáma
DATPTR	0043-0044	67	aktuális DATA mutatója
INPPTR	0045-0046	69	INPUT rutin vektora
VARNAM	0047-0048	71	aktuális BASIC változó neve
VARPNT	0049-004A	73	aktuális BASIC változó értékére mutató

FORPNT	004B-004C	75	mutató: FOR/NEXT segédváltozó
VARTXT	004D-004E	77	mutató a programszámlálónak
OPMASK	004F	79	segédváltozó
DEFPNT	0050-0051	80	mutató az FN definíciójára
DSCPNT	0052-0054	82	
HELPER	0055	85	jelző: HELP vagy LIST használja
JMPER	0056	86	
	0057	87	
OLDOV	0058	88	
DECCNT	005F	95	tizedesjegyek száma
TENEXP	0060	96	
DPTFLG	0061	97	tizedespont jelzője
FAC	0063-0068	99	lebegőpontos akkumulátor#1
SGNFLG	0069	105	hatványsor kiértékeléshez segédregiszter
ARG	006A-006F	106	lebegőpontos akkumulátor#2
ARISGN	0070	107	előjel összehasonlítás eredménye
FACOV	0071	108	FAC#1 kerekítés
FBUFPT	0072-0073	114	mutató: kazetta puffer
AUTOINC	0074-0075	116	AUTO növekménye (=0 nincs)
MVDFLG	0076	118	Jelző: 10K grafikus képernyő allokált-e
ZPTEMP	0077	119	Ideiglenes tároló (MOVSPR, MID$)
HULP	0078	120	számláló
SYNTMP	0079	121	töltésnél segédváltozó
DSDEC	007A-007C	122	DS$ leírója
TOS	007D	125	futási verem teteje
RUNMOD	007F	127	parancs/program mód jelzője
PARSTS	0080	128	DOS kiértékelő jelzője
PARSTX	0081	129	
OLDSTK	0082	130	

COLSEL	0083	131	aktuális szín
MULCO1	0084	132	
MULCO2	0085	133	
FGROUND	0086	134	
SCALE$X	0087-0088	135	
SCALE$Y	0089-008A	137	
STOPNB	008B	139	PAINT megállási jelzője
GRAPNT	008C-008D	140	grafikus rutinhoz cím
VTEMP1	008E	142	grafikus rutinhoz segédregiszter
VTEMP2	008F	143	grafikus rutinhoz segédregiszter

STATUS	0090	144	STATUS byte
STKEY	0091	145	STOP billentyű jelzője
SVXT	0092	146	szalag segédváltozó
VERCK	0093	147	LOAD vagy VERIFY jelző
C3P0	0094	148	a soros kapu pufferénekjelzője
BSOUR	0095	149	a soros kapu puffere
SYNO	0096	150	kazetta szinkronizáció száma
XSAV	0097	151	ideiglenes tároló BASIN-re
LDTND	0098	152	index a logikai file-okhoz
DFLTN	0099	153	default input eszköz
DFLTO	009A	154	default output-eszköz

PRTY	009B	155	kazetta paritás
DPSW	009C	156	kazetta kapcsoló
MSGFLG	009D	157	operációs rendszer üzenet jelzője
PTR1	009E	158	kazetta ideiglenes tároló
PTR2	009F	159	kazetta ideiglenes tároló
TIME	00A0-00A2	160	a 24 órás óra 1/60 másodpercekben
R2D2	00A3	163	soros busz használata: jelző
BSOUR1	00A4	164	ideiglenes változó: soros busz használja
COUNT	00A5	165	ideiglenes változó: soros busz használja
BUFPT	00A6	166	kazetta mutató

INBIT	00A7	167	RS232, kazetta
BITCI	00A8	168	RS232, kazetta: olvasási hiba
RINONE	00A9	169	RS232
RIDATA	00AA	170	RS232, kazetta
RIPRTY	00AB	171	RS232 paritás jelző tárolása
SAL	00AC	172	képernyő görgetés
SAH	00AD	173	
EAL	00AE-00AF	174	kazetta vég címe/ program vége
CMP0	00B0	176	kazetta időzítő konstans
TEMP	00B1	177	
TAPE1	00B2-00B3	178	kazetta puffer címe
BITTS	00B4	180	RS232
NXTBIT	00B5	181	RS232
RODATA	00B6	182	RS232
FNLEN	00B7	183	aktuális file név hossza
LA	00B8	184	aktuális logikai file szám
SA	00B9	185	aktuális másodlagos cím
FA	00BA	186	aktuális hardver szám
FNADR	00BB-0BC	187	az aktuális filenévre mutat
ROPRTY	00BD	189	RS232
FSBLK	00BE	190	kazetta számláló
DRIVE	00BF	191	soros busz puffer
CAS1	00C0	192	kazetta billentyű érzékelés
STA	00C1-00C2	193	I/O kezdőcím
TMP2	00C3-00C4	195	kazetta töltési állandó
DATA	00C5	197	kazetta irás/olvasás adat
BA	00C6	198	az aktuális LOAD/SAVE/VERIFY művelet szelete
FNBANK	00C7	199	az aktuális FN definíció szelete
RIBUF	00C8-00C9	200	RS232 input puffer mutató
ROBUF	00CA-00CB	202	RS232 output puffer mutató

KEYTAB	00CC-00CD	204	billentyűzet definíciók
IMPARM	00CE-00CF	206	
NDX	00D0	208	index a billentyűzet pufferbe
KYNDX	00D1	209	feldolgozatlan billentyű a pufferbe jelzője
KEYIDX	00D2	210	mutató a billentyűzet pufferbe
SHFLAG	00D3	211	SHIFT jelzője
SFDX	00D4	212	aktuális billentyű indexe
LSTX	00D5	213	utolsónak olvasott billentyű kódja
CRSW	00D6	214	<RETURN> jelzője
MODE	00D7	215	40/80 jelzője

GRAPHM	00D8	216	szöveges/grafikus kijelzés jelzője
CHAREN	00D9	217	RAM/ROM VIC karakter jelzője (2. bit)

KEYSIZ	00DA	218	Funkciós billentyűk adatai
KEYLEN	00DB	219	
KEYNUM	00DC	220	
KEYNXT	00DD	221	
KEYBNK	00DE	222	
KEYTMP	00DF	223	

PNT	00E0-00E1	224	mutató az aktuális sorba (szöveg)
USER	00E2-00E3	226	mutató az aktuális sorba (attribútum)
SCBOT	00E4	228	ablak alsó sora
SCTOP	00E5	229	ablak felső sora
SCLF	00E6	230	ablak bal széle
SCRT	00E7	231	ablak jobb széle
LSXP	00E8	232	INPUT kezdő oszlopa
LSTP	00E9	233	INPUT kezdő sora
INDX	00EA	234	INPUT sor vége
TBLX	00EB	235	kurzor sora
PNTR	00EC	236	kurzor oszlopa
LINES	00ED	237	a képernyő maximális sorainak a száma
COLUMNS	00EE	238	a képernyő maximális oszlopainak a száma
DXATAX	00EF	239	a kiírandó karakter
LSTCHR	00F0	240	az utoljára kiírt karakter
COLOR	00F1	241	a kiírandó karakter attribútuma
TCOLOR	00F2	242	elmentett attribútum
RVS	00F3	243	inverz kiírási mód jelzője
QTWS	00F4	244	idézőjel üzemmód jelzője
INSRT	00F5	245	beszúrási üzemmód jelzője
INSFLG	00F6	246	auto-beszúrási üzemmód jelzője
LOCKS	00F7	247	<SHIFT-C=> és <CTRL-S> letiltása
SCROLL	00F8	248	letiltja a képernyő görgetést
BEEPER	00F9	249	letiltja a <CTRL-G>-t
FREKZP	00FA-00FE	250	szabad terület
LOFBUF	00FF	255	

FBUFFR	0100	256	DOS filenév puffer (16 byte)
XCNT	0110	272	DOS ciklusszámláló
DOSF1L	0111	273	DOS filenév 1 hossz
DOSDS1	0112	274	DOS lemezegység 1
DOSF2L	0113	275	DOS filenév 2 hossz
DOSDS2	0114	276	DOS lemezegység 2
DOSF2A	0115-0116	277	DOS filenév 1: cím
DOSOFL	0117-0118	279	BLOAD/BSAVE kezdő cím
DOSOFH	0119-011A	281	BLOAD/BSAVE végcím
DOSLA	011B	283	DOS logikai file szám
DOSFA	011C	284	DOS egységszám
DOSSA	011D	285	DOS másodlagos cím
DOSRCL	011E	286	DOS rekordhossz
DOSBNK	011F	287	DOS szeletszám

DOSDID	0120-0121	288	DOS ID byte-ok
DIDCHK	0122	290	DOS ID jelző
BNR	0123	291	mutató a kezdő számhoz
ENR	0124	292	mutató a végszámhoz
DOLR	0125	293	dollár jele
FLAG	0126	294	vessző jele
SWE	0127	295	számláló
USGN	0128	296	az exponens előjele
UEXP	0129	297	mutató az exponensre
VN	012A	298	a tizedespont előtti jegyek száma
CHSN	012B	299	jobbra igazítás jelzője
VF	012C	300	a tizedespont előtti helyek száma
NF	012D	301	a tizedespont utáni helyek száma
POSP	012E	302	+/- jelzője
FESP	012F	303	exponens mező jelzője
ETOF	0130	304	kapcsoló
CFORM	0131	305	karakter számláló
SNO	0132	306	előjel helye
BLFD	0133	307	üres/csillag jelző
BEGFD	0134	308	mutató a mező kezdetére
LFOR	0135	309	a formátum mező hossza
ENDFD	0136	310	mutató a mező végére
SYSTK	0137-01FF	311	rendszer verem
BUF	0200-02A1	512	rendszer input puffer

FETCH	02A2-02AE	674	LDA(-),Y tetszőleges szeletről
FETVEC	02AA	682	indirekt vektor FETCH-hez
STASH	02AF-02BD	687	STA(-),Y tetszőleges szeletre
STAVEC	02B9	697	indirekt vektor STASH-hez
CMPARE	02BE-02CC	702	CMP(-),Y tetszőleges szelettel
CMPVEC	02C8	712	indirekt vektor CMPARE-hez
JSRFAR	02CD-02E2	716	JSR xxxx tetszőleges szeletre és visszatérés
JMPFAR	02E3-02FB	739	JMP xxxx tetszőleges szeletre

ESCFNVEC	02FC-02FD	764	vektor további függvényekhez
BNKVEC	02FE-02FF	766	vektor funkcionális cartridge-hez
IERROR	0300-0301	768	vektor az X-ben levő hiba kiíratására
IMAIN	0302-0303	770	vektor a rendszer ciklushoz
ICRNCH	0304-0305	772	vektor a tokenizáláshoz
IQPLOP	0306-0307	774	vektor a BASIC szöveg listázásához
IGONE	0308-0309	776	vektor a BASIC karakter értelmezéshez
IEVAL	030A-030B	778	vektor a token kiértékeléshez
IESCLK	030C-030D	780	vektor ESC tokenizáláshoz
IESCPR	030E-030F	782	vektor ESC listázáshoz
IESCEX	0310-0311	784	vektor ESC végrehajtáshoz
ITIME	0312-0313	786	TIME megszakítási vektor
IIRQ	0314-0315	788	IRQ RAM vektor
IBRK	0316-0317	790	BRK RAM vektor
INMI	0318-0319	792	NMI RAM vektor
IOPEN	031A-031B	794	KERNAL: OPEN
ICLOSE	031C-031D	796	KERNAL: CLOSE
ICHKIN	031E-031F	798	KERNAL: CHKIN

ICKOUT	0320-0321	800	KERNAL: CHKOUT
ICLRCH	0322-0323	802	KERNAL: CLRCHN
IBASIN	0324-0325	804	KERNAL: CHRIN
IBSOUT	0326-0327	806	KERNAL: CHROUT
ISTOP	0328-0329	808	KERNAL: STOP
IGETIN	032A-032B	810	KERNAL: GETIN
ICLALL	032C-032D	812	KERNAL: CLALL
EXMON	032E-032F	814	KERNAL: monitor belépési pont
ILOAD	0330-0331	816	KERNAL: LOAD
ISAVE	0332-0333	818	KERNAL: SAVE

CTLVEC	0334-0335	820	Szerkesztő: CTRL végrehajtása
SHFVEC	0336-0337	822	Szerkesztő: SHIFT végrehajtása
ESCVEC	0338-0339	824	Szerkesztő: ESCAPE végrehajtása
KEYVEC	033A-033B	826	Szerkesztő: billentyűzet olvasása
KEYCHK	033C-003D	828	Szerkesztő: billentyűzet tárolása
DECODE	003E-0049	830	vektorok a billentyűzet dekódolására
KEYD	034A-0353	842	billentyűzet puffer
TABMAP	0354-035D	852	TAB-ok jelzése
BITABL	035E-0361	862	a sorok láncolását jelző bitek
LAT	0362-036B	866	logikai file számok
FAT	036C-0375	876	hardver egységszámok
SAT	0376-037F	886	másodlagos címek
CHRGET	0380-039E	896	CHRGET rutin
CHRGOT	0386-039E	902	CHRGOT rutin
QNUM	0390-039E	912	QNUM rutin: Z-bit magas ha $00 vagy $3A C-bit magas ha számjegy

ISUBR0	039F-03D1	927	RAM rutinok
ZERO	03D2-03D4	977	konstans a BASIC-hez
CURR$BANK	03D5	979	SYS, POKE,PEEK szelete
TMPDES	03D6-03D9	980	az INSTR függvény segédregiszterei
FIN$BANK	03DA	984	memóriaszelet a sztring/lebegőpontos szám kon vertáló rutinhoz
SAVSIZ	03DB-03DE	985	ideiglenes változó a SPRSAV-hez
BITS	03DF	989	FAC#1 túlcsordulás jelző
SPRTEMP1	03E0	990	SPRSAV
SPRTEMP2	03E1	991	SPRSAV
FG$BG	03E2	992	előtér/háttér színe
FG$MC1	03E3	993	előtér/több szín#1
VICSCN	0400-07FF	1024	képernyő memória
BRSTACK	0800-09FF	2048	BASIC futási verem

SYS$VEC	0A00-0A01	2560	vektor: BASIC meleg indítás
DEJAVU	0A02	2562	KERNAL inicializáló jelző byte
PALNTS	0A03	2563	PAL/NTS jelzőbyte
INSTAT	0A04	2564	jelző: RESET vagy NMI
MEMSTR	0A05-0A06	2565	mutató a 15-ös szeleten elérhető legalacsonyabb memóriára
MEMSIZ	0A07-0A08	2567	mutató a 15-ös szeleten elérhető legalacsonyabb memóriára

IRQTMP	0A09-0A0A	2569	a kazettás egység ide menti a RAM IRQ vektort
CASTON	0A0B	2571	kazetta
KIKA26	0A0C	2572	kazetta olvasás
STUPID	0A0D	2573	kazetta olvasás
TIMOUT	0A0E	2574	gyors adatátvitel időtúllépés
ENABL	0A0F	2755	RS232 megengedése

M51CTR	0A10	2576	RS232 kontroll regiszter
M51CDR	0A11	2577	RS232 parancs regiszter
M51AJB	0A12-0A13	2578	RS232 felhasználó által definiált sebesség
RSSTAT	0A14	2580	RS232 statusz regiszter
BITNUM	0A15	2581	RS232 elküldendő bitek száma
BAUDOF	0A16-0A17	2582	RS232 teljes idő
RIDBE	0A18	2584	RS232 input puffer mutató: vége
RIDBS	0A19	2585	RS232 input puffer mutató: eleje
RODBE	0A1A	2586	RS232 output puffer mutató: vége
RODBS	0A1B	2587	RS232 output puffer mutató: eleje
SERIAL	0A1C	2588	gyors átvitel jelzője
TIMER	0A1D	2589	az óraregiszter módosító értéke

XMAX	0A20	2592	a billentyűzet puffer hossza
PAUSE	0A21	2593	<CTRL-S> jelzője
RPTFLG	0A22	2594	billentyűzet ismétlésének engedélyezése
KOUNT	0A23	2595	várakozás a billentyűzet ismétlésére
DELAY	0A24	2596	várakozás míg a billentyű lenyomottnak tekinten dő
LSTSHF	0A25	2597	<C=-SHFT> váltás közti várakozás
BLNON	0A26	2598	40 oszlopos képernyő: kurzor mód
BLNSW	0A27	2599	40 oszlopos képernyő: kurzor letiltása
BLNCT	0A28	2600	40 oszlopos képernyő: villogáshoz számláló
GDBLN	0A29	2601	40 oszlopos képernyő: karakter villogás előtt
GDCOL	0A2A	2602	40 oszlopos képernyő: szín villogás előtt
CURMOD	0A2B	2603	80 oszlopos képernyő: kurzor mód
VM1	0A2C	2604	40 oszlopos képernyő: képernyő memória kezdete, felső byte
VM2	0A2D	2605	bittérkép kezdete, felső byte
VM3	0A2E	2606	80 oszlopos képernyő: képernyő memória kezdete, felső byte
VM4	0A2F	2607	80 oszlopos képernyő: attribútum memória kezde te, felső byte
LINTMP	0A30	2608	

SAV80A	0A31	2609	80 oszlopos képernyő
SAV80B	0A32	2610	80 oszlopos képernyő
CURCOL	0A33	2611	80 oszlopos képernyő: kurzor színe villogás előtt
SPLIT	0A34	2612	40 oszlopos osztott képernyő: raszter szám
FNADRX	0A35	2613	X regiszter tárolása szelet műveletek alatt
PALCNT	0A36	2614	PAL számláló
SPEED	0A37	2615	a rendszer sebességének tárolása kazetta és so ros műveletek alatt
SPRITES	0A38	2616	sprite-ok megengedésének tárolása

BLANKING	0A39	2617	
HOLDOFF	0A3A	2618	jelző a VIC chip teljes kontroljára
LDTB1SA	0A3B	2619	a VIC képernyő felső byte-ja képernyő mozgatá sakor
CLREALO	0A3C	2620	8563 karakter feltöltés
CLREAHI	0A3D	2621	8563 karakter feltöltés
	0A40-0A7F	2624	40/80 oszlopos képernyő esetén ideiglenes tároló terület
XCNT	0A80-0A9F	2688	összehasonlításhoz puffer
HULP	0AA0	2720	
FORMAT	0AAA	2730	
LENGTH	0AAB	2731	assembler
MSAL	0AAC-0AAE	2732	assembler
SXREG	0AAF	2735	ideiglenes tárolás
SYREG	0AAF	2736	ideiglenes tárolás
WRAP	0AB0	2737	assembler: ideiglenes tárolás
XSAVE	0AB1	2738	indirekt szubrutinhíváskor X tárolása
DIRECT	0AB3	2739	A T monitor parancs esetén a másolás iránya
COUNT	0AB4	2740	parancsfelismerő (monitor)
NUMBER	0AB5	2741	parancsfelismerő (monitor)
SHIFT	0AB6	2742	parancsfelismerő (monitor)
TEMPS	0AB7-0ABF	2743	

CURBNK	0AC0	2753	ROM száma az aktuális funkcionális cartridge-hez
PAT	0AC1-0AC4	2753	a cartridge-ok fizikai címei
DK$FLAG	0AC5	2757	idegen nyelvű szerkesztő
	0AC6-0AFF	2758	foglalt
TBUFFER	0B00-0BFF	2810	kazetta puffer(192) byte; auto-boot terület
RS232I	0C00-0CFF	3072	RS232 input puffer
RS232O	0D00-0DFF	3328	RS232 output puffer
	0E00-0FFF	3584	sprite definíciós terület
PKYBUF	1000-1009	4096	<F.> billentyűkre definiált szövegek hossza
PKYDEF	100A-10FF	4106	<F.> billentyűkhöz rendelt szövegek

DOSSTR	1100-1130	4352	DOS parancssztring puffer; 48 byte és a hossz
VWORK	1131	4401	grafikus változó
XYPOS	1131-1132	4401	
XPOS	1131-1132	4401	grafika: X koordináta
YPOS	1133-1134	4403	grafika: Y koordináta
XDEST	1135-1136	4405	grafika: X koordináta, tárgy
YDEST	1137-1138	4407	grafika: Y koordináta, tárgy
XYABS	1139-113A	4409	grafika: vonal rajzolásához ugyanazok
XABS	1139-113A	4409	
YABS	113B-113C	4411	
XYSGN	113D-113E	4413	
XSGN	113D-113E	4413	
YSGN	113F-1140	4415	
FCT	1141-1144	4417	faktor
ERRVAL	1145-1146	4421	hiba
LESSER	1147	4423	kisebb végpont
GREATR	1148	4424	nagyobb végpont
ANGSGN	1149	4425	a szög előjele

SINVAL	114A-114B	4426	a szög szinusza
COSVAL	114C-114C	4428	a szög koszinusza
ANGCNT	114E-114F	4430	ideiglenes

XCIRCL	1150-1151	4432	kör középpontja, X koordináta
YCIRCL	1152-1153	4434	kör középpontja, Y koordináta
XRADUS	1154-1155	4436	az ellipszis X irányú féltengelye
YRADUS	1156-1157	4438	az ellipszis Y irányú féltengelye
ROTANG	1158-115B	4440	a forgatás szöge
ANGBEG	115C-115D	4444	kezdő szög
ANGEND	115E-115F	4446	végső szög
XRCOS	1160-1161	4448	X féltengely * COS(forgatás szöge)
YRSIN	1162-1163	4450	Y féltengely * SIN(forgatás szöge)
XRSIN	1164-1165	4452	X féltengely * SIN(forgatás szöge)
YRCOS	1166-1167	4454	Y féltengely * COS(forgatás szöge)

CHRPAG	1168	4456	A CHAR BASIC utasításhoz a karaktergenerátor felső byte-ja
BITCNT	1169	4457	GSHAPE ideiglenes tároló
SCALEM	116A	4458	skála mód jelzője
WIDTH	116B	4459	dupla szélesség jelzője
FILFLG	116C	4460	BOX parancsnál a festés jelzője
BITMSK	116D	4461	ideiglenes tároló
NUMCNT	116E	4462	
TRCFLG	116F	4463	nyomkövetési üzemód jelzője
RENUMT1	1170-1171	4464	RENUMBER használja
RENUMT2	1172-1173	4466	RENUMBER használja
T3	1174	4468	
T4	1175-1176	4469	
VTEMP3	1177	4471	grafika
VTEMP4	1178	4472	
VTEMP5	1179	4473	
ADRAY1	117A-117B	4474	mutató: lebegőpontos -> egész konverzió
ADRAY2	117B-117C	4476	mutató: egész -> lebegőpontos konverzió
SPRITDAT	117E-11D5	4478	sprite sebesség/irány táblázat
VICSAVE	11D6-11EA	4566	VIC 21 regiszterének elmentése
UPLOW	11EB	4587	mutató: kisbetű/nagybetű karakterkészlet a CHAR parancsnak
UPGRAPH	11EC	4588	mutató: nagybetűk/grafikák karakterkészlet a CHAR parancsnak
DOSSA	11ED	4589	
OLDLIN	1200-1201	4608	előző BASIC sor száma
OLDTXT	1202-1203	4610	mutató CONT parancshoz

PUFILL	1204	4612	USING szóköz karakter
PUCOMA	1205	4613	USING ezredes jel
PUDOT	1206	4614	USING tizedes jel
PUMONY	1207	4615	USING pénzjel
ERRNUM	1208	4616	hibarutin: hiba kódszáma
ERRLIN	1209-120A	4617	hibarutin: BASIC sorszám ($FFFF = nincs hiba)
TRAPNO	120B-120C	4619	ON ERROR hibarutin címe ($FFxx = nincs)

TMPTRP	120D	4621	
ERRTXT	120E-120F	4622	mutató: hiba szövege
TEXTTOP	1210-1211	4624	mutató: szöveg tetejére
MAXMEM0	1212-1213	4626	RAM 0-ban BASIC által használható legnagy. cím
TMPTXT	1214-1215	4628	DO-LOOP: ideiglenes tároló
TMPLIN	1216-1217	4630	
USRPOK	1218	4632	USR BASIC függvény: JMP kódja
USRADR	1219-121A	4633	USR címe
RNDX	121B-121F	4635	RND függvény
CIRSEG	1220	4640	kör rajzolásnál a következő pont szögtávolsága
DEJAVU	1221	4641	inicializálás jelzése
TEMPO	1222	4642	
VOICES	1223-1228	4643	
NTIME	1229-122A	4649	
OCTAVE	122B	4651	
SHARP	122C	4652	
PITCH	122D-122E	4653	
VOICE	122F	4655	
WAVE0	1230-1232	4656	
DNOTE	1233	4659	
FLTSAV	1234-1237	4660	
FLTFLG	1238	4664	
NIBBLE	1239	4665	
TONNUM	123A	4666	
TONVAL	123B-123D	4667	
PARCNT	123E	4668	
ATKTAB	123F-1248	4669	
SUSTAM	1249-1252	4681	
WAVTAB	1253-125C	4691	
PULSLW	125D-1266	4701	
PULSHI	1267-1270	4710	
FILTERS	1271-1275	4725	
	1276-1280	4726	megszakítások
	1281-12B2	4737	BASIC hanggenerálás használja
WINDTMP	12B3-12B6	4787	ablak kezelés: ideiglenes
SAVRAM	12B7-12F9	4791	SPRDEF és SAVSPR használja
DEFMOD	12FA	4858	SPRDEF és SAVSPR mód
LINCNT	12FB	4859	SPRDEF és SAVSPR használja
SPNUM	12FC	4860	SPRDEF és SAVSPR használja
IRQWFLG	12FD-12FF	4861	BASIC megszakítás kezelés
	1300-17FF	4864	szabad terület
	1800-1C00	6144	funkcionális ROM-ok számára foglalt terület. Ha nincs, szabad
RAMBOT	1C00	7168	BASIC munkaterület ($1C00-$EFFF)
	1C00	7168	grafika: színmemória; $1C00-$1FFF
	2000	8192	grafika: bit térkép; $2000-$3FFF
	4000	16384	C128 BASIC ROM; grafikánál a BASIC terület eleje ($4000-$EFFF)
	8000	32768	C128 BASIC ROM (második rész)

9. fejezet

A 1570/71-es lemezegység használata

9.1 Bevezetés

A Commodore 128-as mikrogépcsaládhoz a Commodore cég - előző lemezegységeitől teljesen eltérő lemezegységeket fejlesztett ki. Az új, 1570 illetve 1571 típusjelű lemezegységek a Commodore 1541-essel írt lemezegységek olvasásán túl képesek az MFM formátumú lemezek olvasására is. Ezen túlmenően a számítógép és a lemezegység közti adatátvitel két különböző formában történhet. A másik adatátviteli protokol egy sokkal gyorsabb - hardver úton megvalósuló - adatátvitelt biztosít. Az 1570-es és az 1571-es lemezegységek mndenben megegyeznek, kivéve, hogy az 1570-es egyoldalas, míg a 1571-es kétoldalas adatrögzítésre képes.

A 1570/71-es lemezegység csatlakoztatható a Commodore 64, a Plus 4, a Commodore 16 számítógépekhez is, s használható a Commodore 128 C-64-es üzemmódjában. Ezekben az esetekben azonban a lemezegység sajátosságai nem használhatók ki, mert az MFM formátumú lemezek vagy a gyors adatátvitel használatához a Commodore 128 által biztosított hardver lehetőségekre is szükség van (lásd 9.2)!

A 1570/71-es lemezegységek két üzemmódban dolgoznak. Az első 1571-esnek nevezett üzemmódban a lemezegység valamennyi lehetőséget kihasználhatjuk, míg a második, un. 1541-es üzemmódban csak azokat, amelyekre egy 1541-es egység képes.

Attól függően, hogy hogyan és milyen sorrendben kapcsoltuk be a számítógépet és a lemezegységet, a 1571-es lemezegység vagy 1541-es vagy 1571-es üzemmóddal indul el.

a/ Először a lemezegységet kapcsoljuk be, azután a Commodore 128-at. Ebben az esetben a gép C-128-as, a lemezegység 1571-es üzemmódba kerül.

b/ Először a számítógépet kapcsoljuk be, majd miután a rendszer bejelentkezett, a lemezegységet. Ebben az esetben a gép C-128-as, a lemezegység 1541-es üzemmódba kerül. Amikor azonban az első lemezegységnek szóló utasítást elküldjük, az egység automatikusan 1571-es üzemmódba lép át. Ha azt akarjuk, hogy 1541-es üzemmódban maradjon, akkor a

```
PRINT#15,"U0>M0"
```

parancsot kell elküldeni az előzőleg már megnyitott 15-ös logikai számú parancscsatornán. (Részletesen lásd a 9.3 részben.)

c/ Először a lemezegységet kapcsoljuk be, majd a számítógépet, s a rendszer bejelentkezése után kiadjuk a GO64 parancsot. Ezzel a gép C-64-es, a lemezegység 1571-es üzemmódba kerül. Ebben az esetben a 1571-es egység mindkét oldalát tudjuk C-64-es üzemmódban is olvasni, különben a lemezegység pontosan olyan, mintha 1541-es lenne.

d/ Bekapcsoljuk a lemezegységet, majd a <C=> billentyű lenyomva tartása közben bekapcsoljuk a számítógépet. Ezzel a gép C-64-es, a lemezegység 1541-es üzemmódba tér át.

A 1570/71-es lemezegységhez tartozó DEMO lemezen találhatunk egy DOS SHELL elnevezésű programot, amelyik nagy mértékben megkönnyíti a lemezes file-ok kezelését. Erről részletesen a 9.4 részben szólunk.

A 1571-es lemezegységnek van egy óriási hátránya: még 1541-es üzemmódban sem kompatibilis a 1541-es lemezegységgel, így például a Commodore 64/1541-es párosításra kidolgozott gyorsmásolók és töltők legtöbbje nem működik megfelelően. Magam egyetlen másoló programot találtam, amelyiket a Commodore 128-as C-64-es üzemmódjában a 1571-essel is lehetett használni!

A védett lemezek beolvasásával szerencsére nincs probléma. A 1571-es időzítesei - ha 1541-es üzemmódban használjuk - megegyeznek a 1541-es időzítéseivel, ezért a legjobban védett lemezek is probléma nélkül olvashatók segítségével. Ennek ellenére az a véleményem, hogy nem nélkülözhetünk egy igazi 1541-es lemezegységet sem.

9.2 A 1570/71-es hardver felépítése

A 1571-es lemezegység - hasonlóan elődjeihez - Mostek 6502a alapú, s periféria chipnek két 65C22A chipet tartalmaz. Az egyik a lemezegység fizikai vezérlését végzi, a másik segítségével kommunikál a rendszer a számítógéppel. Mindkét lemezegység 32Kbyte ROM-ot és 2Kbyte RAM-ot tartalmaz.

A 1570 egyetlen, míg a 1571 két darab kombinált író-olvasó fejet tartalmaz. Ennek segítségével mind GCR (group code recording) mind MFM (modified frequency modulation) formátumú lemezek olvashatók. A két formátum esetén a lemez tároló kapacitása is eltér.

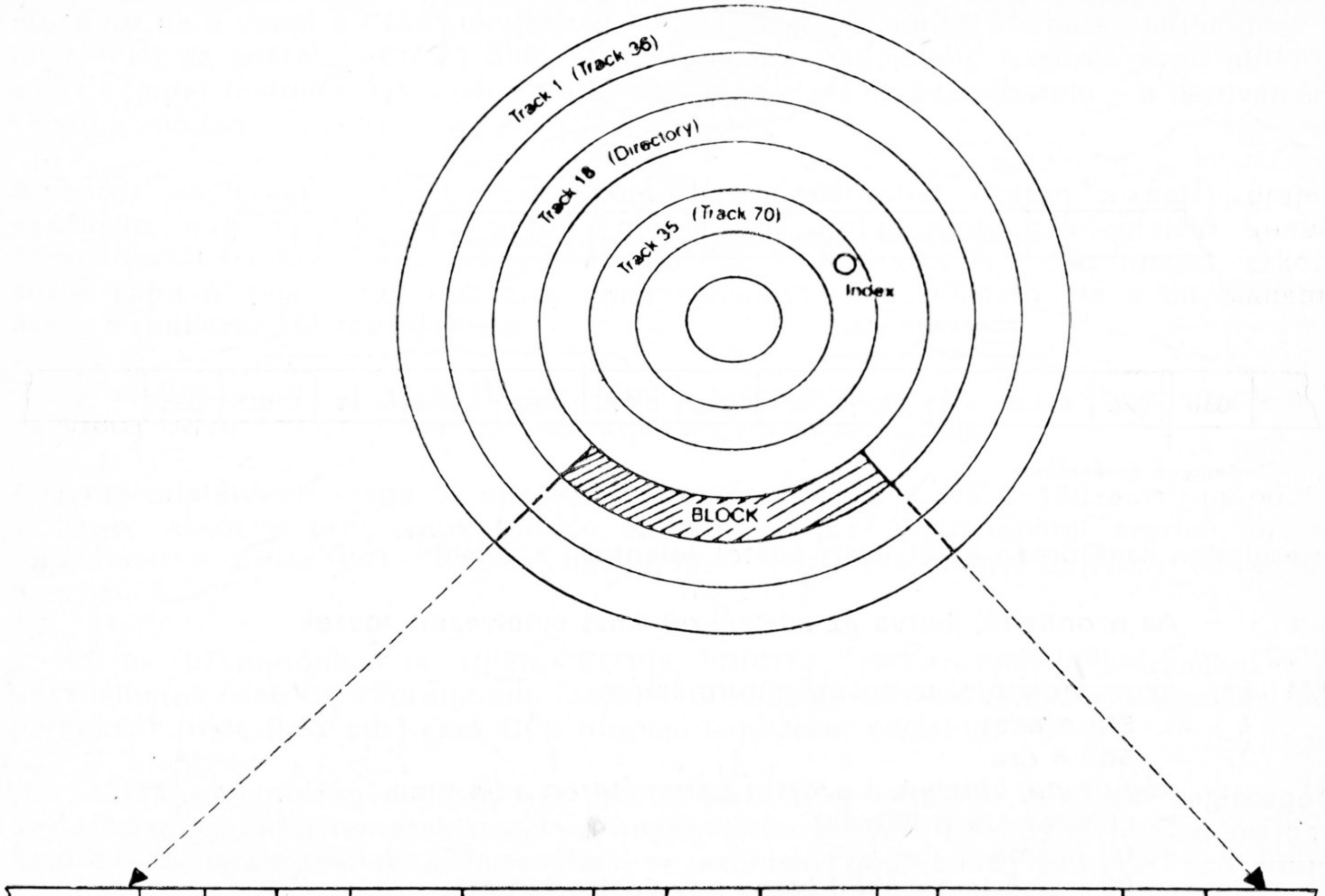
GCR formátum		Lemezegység 1570	1571
Formázatlan kapacitás (byte)		252019	252019*2
Formázott kapacitás (byte)		174848	349696
Legnagyobb SEQ file (byte)		168656	337312
Legnagyobb REL file (byte)		167132	167132
maximális rekordszám		65535	65535
File-ok száma		144	144
Sávok (tracks) száma		35	70
Szektorok sávonként		17-21	17-21
Blokkok száma -	összes	683	1366
	szabad	664	1328
Byte-ok egy blokkban		256	256

MFM formátum

Formázatlan kapacitás	250000 (oldalanként)
Formázott kapacitás	
Szektorméret 128	133120 (oldalanként)
Szektorméret 256	163840 (oldalanként)
Szektorméret 512	184320 (oldalanként)
Szektorméret 1024	204800 (oldalanként)
Sávok (tracks) száma	40 (oldalanként)
Szektorok száma sávonként	
Szektorméret 128	26
Szektorméret 256	16
Szektorméret 512	9
Szektorméret 1024	5

GCR formátumú adattárolás

A GCR formátumot elsősorban a Commodore cég használja a gépeihez gyártott lemezegységek esetén. A 1541/1551/1570/1571-es lemezegységek standard formátumának ez számít. A GCR formátum a sávokat kívülről befelé számozza 1-től 35-ig, illetve kétoldalas egység esetén 36-tól 70-ig. Az adatrögzítés jellemzője, hogy a külső sávokban több blokkot ír mint a belsőben, ezzel is bővítve a tárolási kapacitást. Az egyetlen blokkba elhelyezett adatok - túl az információt hordozó 256 byte-on az alábbiak:

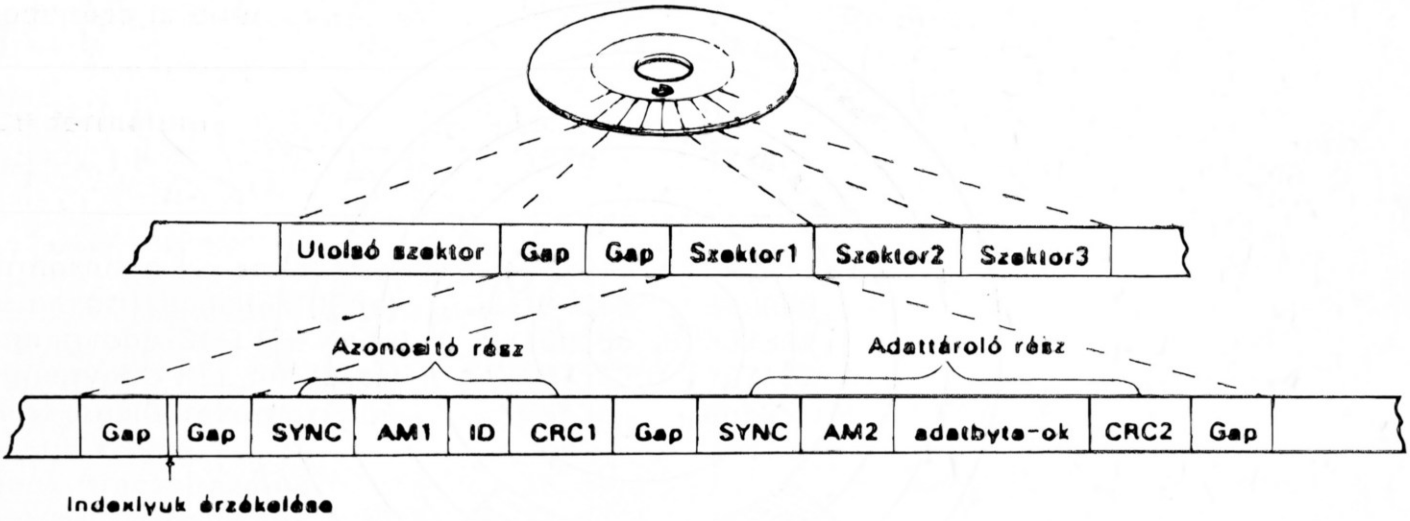


Az egyes részek jelentése az alábbi:

SYNC1 – 40 '1'-es bit (a maximális írási sűrűséggel)
08 – A lemezegység ID-jét azonosító jelsorozat
CS1 – az ID1,ID2,TRACK,SECTOR értékek ellenőrző összege
SECTOR – a szektor száma
TRACK – a blokkot tartalmazó sáv száma
ID1,ID2 – a lemez két karakteres azonosítója, amelynek értéke minden blokkba beíródik
GAP1 – 72 GCR0 bit vagy 9 darab 01010101
SYNC2 – 40 '1'-es bit
07 – az adatmező azonosítója
CS2 – a 256 adatbyte ellenőrző összege
GAP2 – GCR 0-kból álló változó hosszúságú adatrész. Függ a formázás sebességétől. Az első és utolsó blokk közti rész nagysága eltérhet, a többi mindig ugyanaz.

MFM formátumú adattárolás

Az MFM formátumú tárolás több vonatkozásban is eltér a GCR formátumútól. Az MFM formátum esetén a szektor mérete és ezzel együtt az egy sávon levő szektorok száma változhat. (Ezért a szektor bevezető információja sokkal több adatot kell, hogy tartalmazzon!) Az MFM formátum a lemezen levő index lyukat használja az első szektor azonosítására.



Az egyetlen szektorban elhelyezett adatok jelentése az alábbi:

Gap – Az azonosító, illetve az adatrészek közt elhelyezett részek
SYNC – 6 nulla byte
AM1 – az azonosító rész bevető információja
$fe = adat
$c7 = óra
ID – négy byte, amelyik a szektor paramétereit adja meg
= cilinder szám (00-4a)
= fej száma
00 : első lemezoldal (vagy egyoldalas lemez)
01 : második lemezoldal

= szektor sorszáma
01-1A : szektorméret 128
01-0f : szektorméret 256
01-09 : szektorméret 512
01-05 : szektorméret 1024
= szektorméret
00 : 128 byte
01 : 256 byte
02 : 512 byte
03 : 1024 byte

CRC1 - ellenörző összeg
AM2 - az adatmező azonosítója
$fb = adat
$f8 = kontroll mező
CRC2 - ellenörző összeg

MFM formátumú lemezek esetén a sávok számozása 0-tól 39-ig tart. A szektorok számozása - eltérően a GCR formátumtól - 1-gyel kezdődik, s a szektormérettől függően az 5,9,15 vagy 26 értékekig tart.

A gyors adatátvitel

A Commodore 64 számítógép a perifériákkal való kommunikációra három - TTL szintű, szoftverből kezelt vonalat használt - az ATN, a CLK és a DATA vonalakat. A 1570/71-es lemezegységek az SRQ vonalat használják, ami egy hardver irányított gyors órajelként működik. Ez a vonal a CIA#1 órajelével kötődik össze, s ennek ütemére shiftelődnek be (illetve ki) az adatok a CIA#1 chip soros kapujába (kapujából). A soros kapu pufferelt, ezért az adat feldolgozása elvégezhető, miközben a másik adat átvitele - a hardver által vezérelt módon - megtörténik.

A soros kapu működésének jellemzője, hogy adatkivitel esetén a saját órajelét használja, míg adatbevitel esetén kívülről várja az órajelet. A megfelelő vonalak összekötését az MMU 5. regiszterének ($D505) 3. bitje vezérli. Ha ez magas, akkor a soros kapu a belső órát használja, tehát kimenetnek használható. Ha a bit alacsony, akkor a rendszer külső órajelt vár.

C-64-es üzemmódban az MMU chip nincs a rendszerben, ezért - szemben pl. a 80 oszlopos videochippel - nem használhatjuk a gyors adatátvitelt.

A gyors adatátvitelt maga az operációs rendszer használja. Ha C-128-as üzemmódban vagyunk, akkor a programok töltése és mentése az új adatátvitel szerint történik. Hasonlóan a CP/M BIOS rutinok úgy vannak kiképezve, hogy a gyors adatátvitelt használják.

C-128-as üzemmódban a OPEN, CLOSE, INPUT#, GET#, PRINT# parancsok nem használhatók csak GCR formátumú lemezekre. Hasonlóan a közvetlen írási/olvasási DOS parancsok (B-R, B-W stb.) csak GCR formájú lemezeket olvasnak.

A 1570/71-es lemezegység egy sor új DOS parancsot tartalmaz, amelyek segítségével az MFM formátumú lemezek olvasása/írása, illetve a gyors adatátvitel megvalósítható. Ezeknek a parancsoknak a használatához azonban gépi kódú rutinokat kell írnunk, amelyek a gyors adatátvitelt C-128 módban megvalósítják, mert a KERNAL rutinok ilyeneket nem tartalmaznak. Egy lehetséges megvalósítást a 9.5 fejezetben mutatunk be.

9.3 A lemezegység BCIS utasításai

A 1570/71-es lemezegység - a 1541-es lemezegységhez képest - egy új utasításcsoporttal, a BCIS utasításokkal bővül ki. A BCIS (Burst Command Instruction Set) a gyors átviteli parancsokat tartalmazza. Ezen túl ezekkel a parancsokkal írhatjuk/olvashatjuk/formázhatjuk az MFM formátumú lemezeket.

A BCIS parancsokat ugyanúgy kell kiadnunk, mint az összes többi parancsot, tehát a 15-ös csatornán keresztül. A különbség csupán annyi, hogy a legtöbb BCIS parancs kiadása után a lemezegység és a Commodore 128 közt a gyors protokoll szerinti adatcserét kell lebonyolítanunk. S ez nemcsak az adatátvitelre, hanem a lemezegység állapotára vonatkozó információkra is igaz.

Ebben a részben a BCIS utasításkészletét ismertetjük. Alkalmazására egy összetett példát a 9.5 részben mutatunk be.

Burst Read
A parancs segítségével a lemez és a számítógép közt gyors adatátvitelt valósíthatunk meg. A lemezegység a parancsban megadott számú szektort olvas a pufferébe, s küldi el az adatokat - a gyors protokoll szerint - a C-128-nak. Minden egyes szektor átvitele előtt egy status byte-ot küld el a számítógépnek.

Az olvasási parancs kiadása előtt a lemezt be kell léptetni a rendszerbe vagy az INQUIRE DISK vagy a QUERY DISK FORMAT paranccsal!

Byte	Bit 7	6	5	4	3	2	1	0
0	0	1	0	1	0	1	0	1
1	0	0	1	1	0	0	0	0
2	T	E	B	S	0	0	0	N
3	Az olvasandó sáv száma							
4	Az olvasandó szektor száma							
5	Szektorok száma							
6	Következő sáv (opcionális)							

A parancs második byte-ja adja meg az olvasás módját:
T - adatátvitel (1=nincs adatátvitel)
E - hibák figyelmen kívül hagyása (1=a hibás byte-okat is elküldi)
B - csak a bufferba olvas (1=igen)
S - lemezoldal (csak MFM formátum esetén)
N - meghajtó száma. 1570/71 esetén = 0

Példák:
(I) PRINT#15,"U0";CHR$(0);CHR$(1);CHR$(1);CHR(3)
(II) PRINT#15,"U0";CHR$(16);CHR$(1);CHR$(1);CHR(3)

Az (I) példa hatására a lemezegység 1 sávjának 1 szektorától 3 szektor olvasása történik meg. Az olvasást a gyors adatátviteli protokoll szerint kell elvégezni. Az átvitelre került byte-ok száma függ attól, milyen (GCR, MFM) formátumú lemezt használunk.

A (II) példában egy MFM formátumú lemez hátoldalának 1 sávjának 1 szektorától olvasunk 3 szektort. Ha ugyanezt GCR formátummal szeretnénk elvégezni, akkor a

PRINT#15,"U0";CHR$(0);CHR$(36);CHR$(1);CHR(3)

parancsot kell kiadni.

Burst Write
A parancs segítségével a lemez és a számítógép közt gyors adatátvitelt valósíthatunk meg. A lemezegység a parancsban megadott számú szektort kap a C-128-tól - a gyors protokoll szerint - s írja a megadott szektorokba. Minden egyes szektor átvitele után egy status byte-ot küld el a számítógépnek.

Az írási parancs kiadása előtt a lemezt be kell léptetni a rendszerbe vagy az INQUIRE DISK vagy a QUERY DISK FORMAT paranccsal!

Byte	Bit 7	6	5	4	3	2	1	0
0	0	1	0	1	0	1	0	1
1	0	0	1	1	0	0	0	0
2	T	E	B	S	0	0	1	N
3	Az írandó sáv száma							
4	Az írandó szektor száma							
5	Szektorok száma							
6	Következő sáv (opcionális)							

A parancs második byte-ja adja meg az írás módját:
T - adatátvitel (1=nincs adatátvitel)
E - hibák figyelmen kívül hagyása (1=a hibás byte-okat is elküldi)
B - csak a bufferba olvas (1=igen)
S - lemezoldal (csak MFM formátum esetén)
N - meghajtó száma. 1570/71 esetén = 0

Példák:
(I) PRINT#15,"U0";CHR$(2);CHR$(1);CHR$(1);CHR(3)
(II) PRINT#15,"U0";CHR$(18);CHR$(1);CHR$(1);CHR(3)

Az (I) példa hatására a lemezegység 1 sávjának 1 szektorától 3 szektor írása történik meg. Az írást a gyors adatátviteli protokoll szerint kell elvégezni. Az átvitelre került byte-ok száma függ attól, milyen (GCR, MFM) formátumú lemezt használunk.

A (II) példában egy MFM formátumú lemez hátoldalának 1 sávjának 1 szektorától írunk 3 szektort. Ha ugyanezt GCR formátummal szeretnénk elvégezni, akkor a

```
PRINT#15,"U0";CHR$(2);CHR$(36);CHR$(1);CHR(3)
```

parancsot kell kiadni.

INQUIRE DISK

A parancs segítségével lekérdezhetjük, hogy a lemezegységben milyen lemez is van. A parancs egyben arra is alkalmas, hogy segítségével beléptessük a lemezt a rendszerbe.

A parancs kiadása után a lemezegység egyetlen status byte-ot küld – a gyors átviteli protokoll szerint.

	Bit							
Byte	7	6	5	4	3	2	1	0
0	0	1	0	1	0	1	0	1
1	0	0	1	1	0	0	0	0
2	X	X	X	S	0	1	0	N

Az egyes bitek jelentése:

X – érdektelen
S – lemezoldal (csak MFM formátum esetén)
N – meghajtó (1570/71 esetén = 0)

Példák:

(I) PRINT#15, "U0";CHR$(4)

A fenti parancs egyrészt belépteti a lemezt a rendszerbe, másrészt egyetlen status byte-ot küld válaszul, ami a lemezre vonatkozó legszükségesebb információt tartalmazza.

FORMAT MFM

A parancs segítségével lehetőségünk van MFM formátumú lemezek formázására. Az MFM formátumnál elmondottak alapján még ezen belül is különböző lehetőségeink vannak. Ezeket a paraméterek megadásánál magunk is kiválaszthatjuk. Lehetőség van a lemez részleges megformázására is.

A parancs semmilyen output információt sem ad. Ha a megformázott lemezt használni akarjuk, akkor az INQUIRE DISK vagy a QUERY DISK FORMAT paranccsal be kell léptetni a rendszerbe.

Byte	Bit 7	6	5	4	3	2	1	0
0	0	1	0	1	0	1	0	1
1	0	0	1	1	0	0	0	0
2	P	I	D	S	0	1	1	N
3	1	T	logikai kezdő szektor					
4	eltolás (0)							
5	szektorméret(01 = 256 byte)							
6	utolsó sáv (39)							
7	szektorok száma (15)							
8	logikai kezdő sáv (0)							
9	kezdő sáv eltolás (0)							
10	töltő byte ($e5)							
11-?	szektor tábla							

Az egyes paraméterek jelentése a következő:
P – részleges formázás (1=igen)
I – AM1 és AM2 jelek írása (1=igen)
D – kétoldalas lemez (1=igen)
S – oldal kiválasztása
T – szektor tábla mellékelve (1=igen)
N – meghajtó száma

Példák:
(I) PRINT#15, "U0";CHR$(64+4+2)
(II) PRINT#15, "U0";CHR$(64+32+4+2);CHR$(128);CHR$(0);CHR$(02);
CHR$(39);CHR$(9);CHR$(0);CHR$(0);CHR$(DEC("E5"))

Az első példa a lemez formázására a default paramétereket használja. Ennek megfelelően egyoldalas, 256 byte-os szektorokból álló MFM formátumú lemezeket kapunk.

A második példa kétoldalas, 512 byte-os szektorokból álló (sávonként 9 szektor) lemezt formáz meg. Ez a fizikai formátuma az IBM PC-ken használt kétoldalas, 9 szektoros lemezeknek is.

FORMAT GCR
A parancs segítségével GCR formátumra formázhatunk meg egy lemezt. A különbség az, hogy a lemezre nem íródik fel sem a tartalomjegyzék, sem a BAM. Ezért ezek a lemezek a C-128 szokásos utasításaival (DLOAD, DSAVE stb.) nem használhatók!

A parancs semmilyen output információt nem szolgáltat.:

Byte	Bit 7	6	5	4	3	2	1	0
0	0	1	0	1	0	1	0	1
1	0	0	1	1	0	0	0	0
2	X	X	X	X	0	1	1	N
3	0	X	X	X	X	X	X	X
4	ID alsó byte							
5	ID felső byte							

X – érdektelen
N – meghajtó száma (1570/71 esetén = 0)

SECTOR INTERLEAVE

A parancs segítségével a többszektoros READ és WRITE gyorsolvasás/írás eltolását lehet beállítani. A gyorsolvasás/írás sebessége már olyan, hogy nem mindegy, hogy a következő szektor hol található. Egyetlen szektor átvitelét követően például a status byte-ot ki kell értékelni. Ezalatt a lemez továbbforog. Az eltolás értéke azt mondja meg, hogy a 'következő' szektor fizikailag hol is van. Ha az eltolás (interleave) értéke 5, akkor a rendszer egymást követő szektorai a következőek: 0,5,10,15,20,4,9,14,19,3,8,13,18,2,7, 12,17,1. Természetesen ügyelni kell arra, hogy az írás és az olvasás ugyanazon eltolási értékkel történjen.

A parancs segítségével beállítható és lekérdezhető az eltolás értéke. A lekérdezés esetén a lemezegység – a gyors protokoll szerint – egyetlen byte-ot küld válaszul: az eltolás értékét.

Byte	Bit 7	6	5	4	3	2	1	0
0	0	1	0	1	0	1	0	1
1	0	0	1	1	0	0	0	0
2	W	X	X	0	1	0	0	N
3	eltolás							

X – érdektelen
W – 0=beállítás; 1=lekérdezés
N – meghajtó (1570/71 esetén = 0)

Példák:

(I) PRINT#15, "U0";CHR$(8);CHR$(1)

A fenti példa az eltolás értékét 1-re állítja be. Ez azt jelenti, hogy a BURST READ és BURST WRITE paranccsal írt szektorok fizikailag egymást követő szektorokba kerülnek.

QUERY DISK FORMAT

A parancs segítségével lekérdezhetjük a lemezegységben levő lemez egy adott sávjának formátumát. Ennek a parancsnak a segítségével lehetőség van olyan lemezek beléptetésére is, amelyek legkisebb szektorszáma nem 1.

A parancs kiadása után a lemezegység belépteti a lemezt, s – gyors adatátviteli protokoll segítségével – információt küld a lemezről.

Az első byte egy statusbyte. Ebből megállapítható, hogy GCR vagy MFM formátumról van-e szó. Ha MFM formátumú lemez van a meghajtóban, akkor a lemezegység 5 további byte-ot küld. Ezek jelentése a következő:

1. byte – szektorok száma (a parancsban megadott sávon)
2. byte – logikai sávok száma
3. byte – minimális szektorszám
4. byte – maximális szektorszám
5. byte – CP/M eltolás (hardver eltolás az adott sávon)

Byte	Bit 7	6	5	4	3	2	1	0
0	0	1	0	1	0	1	0	1
1	0	0	1	1	0	0	0	0
2	F	X	X	S	1	0	1	N
3	sávszám (opcionális)							

X – érdektelen
S – oldal (csak MFM esetén)
N – meghajtó száma (1570/71 esetén 0)
F – ha 1, akkor a sávszámnak megfelelő sávon nézi a rendszer az adott információt.

Példa:

(I) PRINT#15, "U0";CHR$(8+2)

Az (I) parancs hatására – gyors adatátviteli protokollal – kiolvashatóak a lemezegységben levő lemez adatai.

INQUIRE STATUS

A parancs segítségével lekérdezhetjük/beállíthatjuk a gyors adatátvitelhez tartozó status byte-ot. A status byte egyes bitjeinek a jelentése a következő:

Bit	7	6	5	4	3	2	1	0
	MODE	DN	méret		állapot			

A 7. bit az MFM formátum jelzője. Ha ez a bit magas, akkor a lemez formátuma MFM, ha alacsony (=0) akkor GCR.

A 6. bit a meghajtó számát jelenti, amire az információ vonatkozik. 1570/71-es egységek esetén ez 0. (Kivéve, ha tévedésből az 1-es meghajtóra hivatkoztunk, s ezért hibaüzenetet kaptunk.)

Az 5. és 4. bitek a szektor méretét tartalmazzák az alábbiak szerint:

00 : 128 byte
01 : 256 byte
10 : 512 byte
11 : 1024 byte

A 3.-0. bitek a hibaüzenet kódját tartalmazzák. Ezek jelentése GCR és MFM formátumú lemezek esetén némileg eltér:

3210	GCR	MFM
000X	Nincs hiba	Nincs hiba
0010	Szektor nincs	Szektor nincs
0011	Nincs SYNC	Nincs AM1 vagy AM2
0100	Adat blokk nincs	-
0101	Adat blokk ellenőrző összeg hibás	Adat blokk CRC összeg hibás
0110	Formázási hiba	Formázási hiba
0111	Ellenőrzési hiba	Ellenőrzési hiba
1000	Irásvédő rés takarva	Irásvédő rés takarva
1001	Azonosító rész ellenőrző összeg hiba	Azonosító rész CRC hiba
1010	Az adat belelóg a következő blokkba	-
1011	ID hiba/lemezcsere	Lemezcsere
1100	-	-
1101	-	-
1110	Szintaktikus hiba	Szintaktikus hiba
1111	Nem létező meghajtó	Nem létező meghajtó

FASTLOAD
A parancs segítségével a gyors adatátvitellel tölthetünk be a memóriába egy PRG vagy SEQ típusú file-t.

A parancs kiadása után a file szektorait a lemezegység - megfelelő sorrendben - elküldi. Az adatátvitel a gyors adatátviteli protokoll szerint történik, s minden szektort egy status byte elküldése előzi meg.

Byte	Bit 7	6	5	4	3	2	1	0
0	0	1	0	1	0	1	0	1
1	0	0	1	1	0	0	0	0
2	P	X	X	1	1	1	1	1
3-?	filenév							

P – nem kötelezően program file (0=csak program file lehet)
X – érdektelen

Példa:

(I) PRINT#15, "U0";CHR$(128+16+8+4+2+1);"ADATOK"

A parancs hatására a gyors protokoll szerint sorban olvashatjuk az "ADATOK" nevű file byte-jait. A file nem kell, hogy PRG típusú legyen. A lemezegység a szektorok első két byte-ján talált sáv-szektor információ alapján azonosítja a file blokkjait, s annak megfelelően küldi el.

USER parancsok
A 1570/71-es lemezegységek további USER utasításokat tartalmaznak, amelyek a GCR (szokásos) formátumú lemezek használatát könnyítik. Ezek a következők:

DOS eltolás beállítása
Szintaxis: "U0>S";CHR$(eltolás)

A parancs hatására a DOS a file-ok felírásánál az általunk megadott 'eltolás' értéket használja.

DOS kísérletek száma
Szintaxis: "U0>R";CHR$(kisérletek)

A DOS – ha nem tud megnyitni egy file-t, vagy hibát észlel írás/olvasás közben – újra kísérletezik. Ezen kísérletek számát állíthatjuk be ezzel a paranccsal.

ROM ellenőrzése
Szintaxis: "U0>T"

A DOS ellenőrző összeget készít a teljes 32K-s ROM-ról, s ezt összehasonlítja a benne tárolt adattal. Ha eltérést talál, a kijelzőn a lámpa 4-et villan. Ha minden rendben van, akkor nem történik semmi.

Működési mód kiválasztása
Szintaxis:
"U0>M1" (1571-es üzemmód)
"U0>M0" (1541-es üzemmód)

Említettük, hogy a 1570/71-es lemezegységek két üzemmódban tudnak működni. A bekapcsoláskor valamelyik üzemmód automatikusan kiválasztódik. Ezen a fenti parancsok kiadásával lehet változtatni.

Oldal kiválasztása
Szintaxis:
"U0>H0" (felső oldal)
"U0>H1" (alsó oldal)

A 1571-es lemezegység lehetőséget nyújt arra, hogy a lemezt két egyoldalas lemezként használjuk. Ebben az esetben meg kell mondani, hogy melyik oldalát használjuk.

Tegyük fel, hogy egy egyoldalas lemezünk van. Üres meghajtó mellett adjuk ki az "U0>H1" parancsot, majd tegyük be az egyoldalas lemezt! Ezután a HEADER parancs segítségével megformázhatjuk a lemez alsó oldalát, a fenti érintetlen marad! Valójában két egyoldalas lemezünk lesz, amelyek közt a fenti parancsok segítségével tudunk átkapcsolni.

Ez főleg olyan kész programok esetén jók, amelyek saját lemezformátumot használnak. Ilyenek pl. a FORTH rendszerek. A lemez felső oldalán tárolhatjuk a rendszerprogramokat, az alsón az adatokat. A módszerhez terrmészetesen az kell, hogy a rendszerből kiadhassunk DOS parancsokat. Ezt azonban a legtöbb rendszer megengedi.

9.4 A DOS SHELL használata

A 1570/71-es lemezegységekkel együtt adnak egy DEMO lemezt is, amelyik tartalmazza a DOS SHELL nevű programot. A DOS SHELL rezidens bővítése a C-128-as üzemmódnak, s lehetővé teszi a lemezen levő file-ok kezelését.

A DEMO lemez önindítású lemez, ha tehát a Commodore 128 bekapcsolása előtt behelyezzük a már bekapcsolt lemezegység meghajtójába, s csak ezután kapcsoljuk be a Commodore 128-at, akkor automatikusan betöltődik és elindul. A DOS SHELL inicializálása után automatikusan visszatér a BASIC-be. Ha szükségünk van rá, akkor az F1 billentyűt kell megnyomnunk. (A billentyűre eredetileg definiált GRAPHICS már nem használható). Ha az F1-et valamiért át kell definiálnunk, akkor a BANK 12: SYS 6656 parancs kiadásával indíthatjuk el.

A DOS SHELL az első aktivizálásakor megkérdezi, hogy milyen nyelven szertnénk vele kommunikálni. A lehetséges választások: angol, francia, német és olasz. A képernyőn, az adott nyelven látszik a kérdés, majd mintegy 6 másodperc múlva újabb nyelven jelenik meg. Ha megnyomjuk a <Szóköz> billentyűt, akkor a kiírt nyelven fognak a menűk és egyéb üzenetek megjelenni. A 6 másodpercet nem kell kivárni, hanem a <CRSR LE> billentyű segítségevel áttérhetünk a következő nyelvre.

A nyelv kiválasztása után a DOS SHELL főmenüje jelenik meg. A kurzor vezérlő billentyűk segítségével a megfelelő funkcióra lépünk, s azt a <Szóköz> billentyű megnyomásával aktivizáljuk. Ha az <F1> billentyűt nyomjuk meg, akkor visszatérünk a BASIC-ba.

DISK/PRINTER SETUP
(lemezegység/nyomtató beállítása)
A funkció lehetőséget biztosít a használt lemezegység és nyomtató karakterisztikájának a kiválasztásához. A DOS SHELL két - A és B jelű lemezegységet használ. Az egyik a 8-as a másik a 8-as vagy 9-es hardver számú egység lehet. Mindkét esetben lehetőség van a meghajtó kiválasztására is. Ha az egyik egységet a 9-esnek definiáljuk, akkor a DOS SHELL szoftver úton átállítja a hardver számát.
A kurzor billentyűk segítségével a megváltoztani kívánt paraméterre kell állnunk, majd megnyomni az <F7> billentyűt. Erre a paraméter értéke megváltozik. Ha mégsem akarjuk megváltoztatni, akkor a <STOP> megnyomásával visszaállíthatjuk az eredeti értéket.

Ha az A és B logikai egységek nem ugyanannak a lemezegységnek ugyanazon meghajtójára vonatkoznak, akkor a többi funkció esetén a rendszer mindig megkérdezi, hogy melyik egységet: az A-t, vagy a B-t használja. A kurzort a használni kívánt egység betűjére kell vinni, majd megnyomni az <F7> billentyűt. Ezután a művelet az így kiválasztott egységre fog végrehajtódni.

RUN A PROGRAM
A funkció lehetővé teszi, hogy a lemezen levő adott programot futtassunk. A funkció kiválasztása a RUN <filenév> BASIC parancs kiadásával ekvivalens.

A funkció kiválasztása után a képernyőn a PRG típusú file-ok listája jelenik meg. A kurzort a megfelelő file-ra kell pozicionálnunk, majd a <Szóköz> billentyű megnyomásával kiválasztani a file-t. Ha meggondotuk magunkat az <F5> billentyű megnyomásával újra választhatunk. A kiválsztott PRG file az <F7> billentyű megnyomása után betöltődik a memóriába és elindul.

FORMAT A DISK

A funkció segítségével lemezeket formázhatunk meg. A DOS SHELL egy üres lemezt kér a meghajtóba, majd megnézi, nincs-e már megformázva. Ha egy formázott lemezt helyeztünk be, akkor kiírja a nevét és azonosítóját, s megkérdezi, hogy tényleg meg akarjuk-e formázni. Ha a kurzort a <Y>-ra visszük, s megnyomjuk a <Szóköz> billentyűt, a formázás megkezdődik.

Ennek első lépéseként a program kéri a lemez nevét és két karakteres azonosítóját. Ezután megformázza a lemezt.

Ha egy régi lemezt formázunk újra, s nem írjuk be a két karakteres azonosítót, akkor a program a régit használja. Ha újonnan megformázott lemezről van szó, akkor véletlenszerűen generál egyet. Ha magunk akarjuk megadni, akkor a lemez neve után vesszőt kell tennünk, majd beírni a két karaktert.

CLEANUP A DISK

A funkció hatására egy VALIDATE parancs hajtódik végre a lemezen. Ha közvetlen írású file volt a lemezen, annak tartalma elvész!

COPY A DISK

A funkció segítségével egy teljes lemezt másolhatunk le. A másolás lefolyása attól függ, hogyan állítottuk be az A és B logikai egység parametereit.

Ha az A és B logikai egység ugyanarra a fizikai meghajtóra vonatkozik, akkor először egy üzenetet kapunk, hogy a memóriában levő program elvész. Ha megnyomjuk az <F5> vagy <STOP> billentyűket, akkor a másolás félbeszakad, s a programunk nem törlődik. Ha folytatni akarjuk a másolást, nyomjuk meg a <Szóköz> billentyűt. Ezután a rendszer felváltva kéri az eredeti (original) és az új (copy) lemezeket a meghajtóba. A megfelelő behelyezése után a <Szóköz> billentyűt kell megnyomni.

Ha másolás közben a <STOP> billentyűt megnyomjuk a másolás félbeszakad s a program visszatér a főmenühöz.

Abban az esetben, ha az A és B fizikailag különböző meghajtók a program megkérdezi, hogy melyikről akarunk másolni, majd felkéri az eredeti és a másolat lemezt. Mindegyik behelyezése után a <Szóköz> billentyűt kell megnyomni. A másolás közben a programunk megmarad.

COPY FILES

A funkció segítségével megadott file-okat másolhatunk egyik lemezről a másikra. A másolandó file-ok kiválasztása a kurzor névre történő pozícionálásával és a <Szóköz> billentyű megnyomásával végezhető el. A kiválasztott filenév törölhető, ha ráállunk és megnyomjuk az <F5> billentyűt. A másolás folyamata bármikor megszakítható a <STOP> billentyű megnyomásával. Ha valamennyi másolandó file-t kiválasztottuk,

akkor a másolás az <F7> billentyű megnyomásával megkezdhető. A biztonság kedvéért a program még visszakérdez, hogy rendben van-e a lista. Ha nyugtázó választ adunk, akkor megkezdődik a másolás, ha nem, akkor folytathatjuk a file-ok kiválasztását.

A másolás végrehajtásának módja megint attól függ, hogy az A és B lemezek ugyanazt a fizikai meghajtót jelentik-e vagy sem. Ha igen, akkor figyelmeztető üzenetet kapunk, hogy a memóriában levő progam elvész. Ezt a <Szóköz> billentyű megnyomásával nyugtázhatjuk, vagy az <F5> megnyomásával megállíthajuk a másolás folyamatát. A program felváltva kéri a meghajtóba az eredeti (original) és a másolat (copy) lemezt. A másolat első bekérése után megnézi, hogy meg van-e formázva. Ha nincs, akkor megformázza. Ha már formázott lemezt használunk, akkor kiirja a lemez nevét, s megkérdezi, hogy nem akarjuk-e megformázni. A <Y> (igen) vagy <N> kiválasztása után a <SZóköz> billentyű megnyomására kezdődik a másolás. Ha a formázást választottuk, akkor megformázza a lemezt. Ha nem, akkor a meglevő file-okhoz írja hozzá az átmásolt file-okat.

Ha az A és B lemezek különböző meghajtóra vonatkoznak, akkor a program nem vész el. A rendszer az egyik, illetve a másik meghajtóba felkéri az eredeti és a masolat lemezt, majd hasonlóan az előző esethez, ellenőrzi, hogy a másolat lemezt megformáztuk-e már. Ha igen, akkor megkerdezi, hogy újra formázza-e. Ezután a másolás végrehajtódik.

DELETE FILES

A funkció segítségével a lemez katalógusából kiválasztott file-okat törölhetjük. A kiválasztás úgy történik, hogy a kurzort a törölni kívánt file-ra pozicionáljuk, majd megnyomjuk a <Szóköz> billentyűt. Hasonlóan vonhatjuk vissza a kiválasztást, csak akkor az <F5> billentyűt kell megnyomnunk. A kiválasztás befejezése után az <F7> megnyomásával kezdhetjük el a törlést. A program visszakérdez, hogy rendben van-e a lista. Az <Y> (igen) vagy <N> (nem) betűre állunk, s megnyomjuk a szóköz billentyűt. Ha nemmel válaszolunk, akkor folytathatjuk a törlendő file-ok kiválasztását, ha igennel, a törlésük megtörténik.

RESTORE FILES

A funkció segítségével törölt file-okat állíthatunk vissza. Ezek kiválasztása hasonlóan történik, mint az előző funkcióban. A visszakérdezés után a program megkisérli a file-okat visszaállítani. Ha ez nem sikerül, akkor hibaüzenetet kapunk, feltüntetve annak a file-nak a nevet, amelyet nem sikerült visszaállítani.

RENAME FILES

A funkció segítségével file-okat nevezhetünk át. A listában a kurzort az átnevezni kívánt file-ra állítjuk, s megnyomjuk a <Szóköz> billentyűt. Ezekután lehetőségünk van az új név megadására. Ha olyan nevet adunk meg, ami már létezik, akkor hibaüzenetet kapunk. Ha mégsem akarunk új file nevet megadni, akkor az <F5> billentyűt kell megnyomnunk.

A szükséges változtatások elvegzése után nyomjuk meg az <F7> billentyűt. A rendszer visszakérdez, hogy rendben vannak-e a változtatások. Ha az <Y>-ra állunk és megnyomjuk a <Szóköz> billentyűt, akkor a program az általunk végzett átnevezéseket visszaírja a lemezre.

REORDER DIRECTORY

A funkció lehetőséget biztosít a katalógus átrendezésére, azaz, arra, hogy a filenevek és a hozzájuk tartozó egyéb bejegyzesek más sorrendben legyenek a katalógusban, mint jelenleg.

Az átrendezést a képernyőn megjelenő listán kell először elvégezni. A kurzort arra a filenévre visszük, amelynek a helyét meg akarjuk változtani, majd megnyomjuk a <Szóköz> billentyűt. Ezután az új helyére visszük és újból megnyomjuk a <Szóköz> billentyűt. Ha minden file-t a helyére raktunk, akkor megnyomjuk az <F7> billentyűt.

A rendszer visszakérdez, hogy rendben van-e az átrendezés. Ha igennel válaszolunk, akkor az új katalógust visszírja a lemezre, ha nemmel, akkor tovább folytathatjuk a rendezést.

9.5 Példák

Ebben a részben két BASIC programot mutatunk be, amelyek ugyanazt a gépi kódú programot használják. A rutin a gyors adatátvitelt valósítja meg egy 1570/71-es lemezegység és a Commodore 128 között. Az MMU chip használata miatt - sajnos - a rutin C-64-es üzemmódban nem használható.

A gépi kódú rész két rutinból áll: a $0C03 alatti az író, a $0C00 alatti az olvasó rutin. Meghívásuk előtt az alábbi memóriarészekbe a megfelelő értékeket be kell írnunk:

$0C06 = szektorméret	1 = 256 byte	
	2 = 512 byte	
	4 = 1024 byte	

<u>Olvasás</u>

$0C09 = a 4. bit magas(1) = a lemez hátoldala
alacsony(0) = a lemez felső oldala
$0C0A = olvasandó sáv száma
$0C0B = olvasandó szektor száma
$0C0C = az olvasni kívánt szektorok száma
$0C0D = következő sáv

<u>Írás</u>

$0C10 = a 4. bit magas(1) = a lemez hátoldala
alacsony(0) = a lemez felső oldala
$0C11 = írandó sáv száma
$0C12 = írandó szektor száma
$0C13 = az írni kívánt szektorok száma

A gépi kódú program a $0400 címtől terjedő memórirészt használja puffernek, ide ír, s innen olvas. Ez jól nyomonkövethetővé teszi az írás-olvasás folyamatát. Normál használat esetén a puffert természetesen máshol kell kijelölni.

Az első BASIC program normál GCR formátumú lemez 0. sávjának egymást követő három szektorába ír 256-256 byte-ot, a byte-ok értéket 0-tól 255-ig változtatva. Az adatok visszaolvasása akármelyik billentyű megnyomásával elkezdődik. Vizuálisan is ellenőrizhető, hogy ugyanazokat az adatokat olvastuk vissza.

A BASIC program a gépi kódú programot DATA sorokban tartalmazza. Az 1040–1070 sorok a rutint a helyére töltik. Ha rosszul gépeltük a DATA sorokat, a 1110. sorban hibaüzenetet kapunk. Ezután az 1150–1190 sorok feltöltik a puffert.

Az előbbiek FAST üzemmódban történnek. (Ha 80 oszlopos képernyőt használunk, akkor a FAST és SLOW utasításokat hagyjuk ki!) Az 1230–1250 sorok beléptetik a lemezt a rendszerbe. Az 1270-ik sor kiírja a három szektort a lemez első sávjára. (A példához megformázott, de másra nem használt lemezt használjunk, mert különben a lemezen levő file-ok megsérülhetnek.)

Az 1290-ik sor visszaolvassa a kiírt szektorokat.

```
1000 fast
1010 :
1020 bank 15: c=dec("0c00"): read a
1030 :
1040 do until a=-1
1050 s=s+a
1060 poke c,a: c=c+1: read a
1070 loop
1080 :
1090 if s<>51811 then begin
1100 slow: scnclr
1110 print "hibas adat a data sorokban!"
1120 end
1130 bend
1140 :
1150 for i=0 to 2
1160 :  for j=0 to 255
1170 : :  poke dec("0400")+256*i+j,j
1180 :  next j
1190 next i
1200 :
1210 slow
1220 :
1230 open 15,8,15
1240 print#15, "u0";chr$(4)
1250 close15
1260 :
1270 sys dec("0c03"): getkey a$
1280 print"<CLR><21*CRSR-LE>";
1290 sys dec("0c00")
1300 :
1310 data 76,21,12,76,189,12,1,85
1320 data 48,0,1,1,3,2,85,48
1330 data 2,1,1,3,2,169,0,133
1340 data 251,169,4,133,252,169,15,162
1350 data 8,160,15,32,186,255,169,0
1360 data 32,189,255,32,192,255,169,0
1370 data 133,250,173,28,10,41,191,141
1380 data 28,10,162,15,32,201,255,162
1390 data 0,160,7,189,7,12,32,210
1400 data 255,232,136,208,246,32,204,255
1410 data 44,28,10,80,96,120,44,13
```

```
1420 data 220,174,12,12,173,0,221,73
1430 data 16,141,0,221,169,8,44,13
1440 data 220,240,251,173,0,221,73,16
1450 data 141,0,221,173,12,220,133,250
1460 data 41,15,201,2,176,55,160,0
1470 data 173,6,12,133,254,169,8,44
1480 data 13,220,240,251,173,0,221,73
1490 data 16,141,0,221,173,12,220,145
1500 data 251,200,208,233,198,254,240,5
1510 data 230,252,76,133,12,202,240,5
1520 data 230,252,76,100,12,88,24,169
1530 data 15,32,195,255,96,88,56,169
1540 data 15,32,195,255,96,169,0,133
1550 data 251,169,4,133,252,169,15,162
1560 data 8,160,15,32,186,255,169,0
1570 data 32,189,255,32,192,255,169,0
1580 data 133,250,173,28,10,41,191,141
1590 data 28,10,162,15,32,201,255,162
1600 data 0,160,7,189,14,12,32,210
1610 data 255,232,136,208,246,32,204,255
1620 data 44,28,10,80,110,120,169,64
1630 data 133,253,160,0,174,19,12,32
1640 data 115,13,173,6,12,133,254,173
1650 data 0,221,205,0,221,208,248,69
1660 data 253,41,64,240,242,165,253,73
1670 data 64,133,253,177,251,141,12,220
1680 data 169,8,44,13,220,240,251,200
1690 data 208,221,198,254,240,5,230,252
1700 data 76,15,13,32,152,13,44,13
1710 data 220,32,171,13,169,8,44,13
1720 data 220,240,251,173,12,220,133,250
1730 data 32,180,13,165,250,41,15,201
1740 data 2,176,16,202,240,5,230,252
1750 data 76,7,13,88,24,169,15,32
1760 data 195,255,96,88,169,15,32,195
1770 data 255,56,96,173,5,213,9,8
1780 data 141,5,213,169,127,141,13,220
1790 data 169,0,141,5,220,169,4,141
1800 data 4,220,173,14,220,41,128,9
1810 data 85,141,14,220,44,13,220,96
1820 data 173,14,220,41,128,9,8,141
1830 data 14,220,173,5,213,41,247,141
1840 data 5,213,96,173,0,221,9,16
1850 data 141,0,221,96,173,0,221,41
1860 data 239,141,0,221,96,-1
```

A második program lehetővé teszi egy tetszőleges MFM formátumú lemez felső oldalán levő szektorainak vizsgálatát. A program először megkérdezi, hogy hányszor 256 byte-ból áll a lemez egy szektora, majd folyamatosan kéri a sáv és a szektor számát. A szektor beolvasás után a képernyőmemóriába kerül. Ezzel a rutinnal MFM formátumú lemezeket is olvashatunk. Ha pl. van egy IBM PC-n használt lemezünk, annak szektorait ezzel a programmal le tudjuk olvasni, s erre építve készíthetünk egy GCR – MFM másolóprogramot is. Sok szerencsét hozzá!

```
1000 fast
1010 :
1020 c=dec("0c00"): read a: bank 15
1030 :
1040 do until a=-1
1050 s=s+a
1060 poke c,a: c=c+1: read a
1070 loop
1080 :
1090 slow
1100 if s<>51811 then begin
1110 scnclr
1120 print "hibas adat a data sorokban!"
1130 end
1140 bend
1150 :
1160 input "sector/256";s
1170 poke dec("0c06"),s
1180 poke dec("0c0c"),1
1190 :
1200 input "track,sector=";x,y
1210 if x=-1 then end
1220 poke dec("0c0a"),x
1230 poke dec("0c0b"),y
1240 :
1250 open 15,8,15
1260 print#15, "u0";chr$(4)
1270 close15
1280 :
1290 print"<CLR><21*CRSR-LE>";
1300 sys dec("0c00")
1310 print ds$
1320 goto 1200
1330 :
1340 data 76,21,12,76,189,12,1,85
1350 data 48,0,1,1,3,2,85,48
1360 data 2,1,1,3,2,169,0,133
1370 data 251,169,4,133,252,169,15,162
1380 data 8,160,15,32,186,255,169,0
1390 data 32,189,255,32,192,255,169,0
1400 data 133,250,173,28,10,41,191,141
1410 data 28,10,162,15,32,201,255,162
1420 data 0,160,7,189,7,12,32,210
1430 data 255,232,136,208,246,32,204,255
1440 data 44,28,10,80,96,120,44,13
1450 data 220,174,12,12,173,0,221,73
1460 data 16,141,0,221,169,8,44,13
1470 data 220,240,251,173,0,221,73,16
1480 data 141,0,221,173,12,220,133,250
1490 data 41,15,201,2,176,55,160,0
1500 data 173,6,12,133,254,169,8,44
1510 data 13,220,240,251,173,0,221,73
1520 data 16,141,0,221,173,12,220,145
1530 data 251,200,208,233,198,254,240,5
```

```
1540 data 230,252,76,133,12,202,240,5
1550 data 230,252,76,100,12,88,24,169
1560 data 15,32,195,255,96,88,56,169
1570 data 15,32,195,255,96,169,0,133
1580 data 251,169,4,133,252,169,15,162
1590 data 8,160,15,32,186,255,169,0
1600 data 32,189,255,32,192,255,169,0
1610 data 133,250,173,28,10,41,191,141
1620 data 28,10,162,15,32,201,255,162
1630 data 0,160,7,189,14,12,32,210
1640 data 255,232,136,208,246,32,204,255
1650 data 44,28,10,80,110,120,169,64
1660 data 133,253,160,0,174,19,12,32
1670 data 115,13,173,6,12,133,254,173
1680 data 0,221,205,0,221,208,248,69
1690 data 253,41,64,240,242,165,253,73
1700 data 64,133,253,177,251,141,12,220
1710 data 169,8,44,13,220,240,251,200
1720 data 208,221,198,254,240,5,230,252
1730 data 76,15,13,32,152,13,44,13
1740 data 220,32,171,13,169,8,44,13
1750 data 220,240,251,173,12,220,133,250
1760 data 32,180,13,165,250,41,15,201
1770 data 2,176,16,202,240,5,230,252
1780 data 76,7,13,88,24,169,15,32
1790 data 195,255,96,88,169,15,32,195
1800 data 255,56,96,173,5,213,9,8
1810 data 141,5,213,169,127,141,13,220
1820 data 169,0,141,5,220,169,4,141
1830 data 4,220,173,14,220,41,128,9
1840 data 85,141,14,220,44,13,220,96
1850 data 173,14,220,41,128,9,8,141
1860 data 14,220,173,5,213,41,247,141
1870 data 5,213,96,173,0,221,9,16
1880 data 141,0,221,96,173,0,221,41
1890 data 239,141,0,221,96,-1
```

A gépi kódú rész listája az alábbi:

```
setlfs    = $ffba
setnam    = $ffbd
open      = $ffc0
close     = $ffc3
chkout    = $ffc9
clrchn    = $ffcc
bsout     = $ffd2
;
mmureg    = $d505
d1timl    = $dc04
d1timh    = $dc05
d1cra     = $dc0e
d1sdr     = $dc0c
d1icr     = $dc0d
d2pra     = $dd00
```

```
;
serial      = $0a1c
;
oldclk      = $fd
stat        = $fa
buffad      = $0400
buffer      = $fb
sector      = $fe
clkin       = %01000000
ckinl       = %10111111
clkout      = %00010000
;
*           = $0C00              ; a program az RS232-es pufferekben van
;
            jmp readsp           ; beolvasó rutin többi része
            jmp writsp           ; kiíró rutin többi része
;
scsize      .byte 1              ; szektorméret (alapértelmezés = 256)
cmobuf      .byte 'u0',$00,1,1,n,2
cmibuf      .byte 'u0',$02,1,1,n,2
cmdlg       = 7                  ; a parancs hossza
;
readsp      lda #<buffad         ; puffer kezdőcímének beállítása
            sta buffer
            lda #>buffad
            sta buffer+1
            ;
            lda #15              ; OPEN 15,8,15
            ldx #8
            ldy #15
            jsr setlfs           ; a csatorna kijelölése
            ;
            lda #0
            jsr setnam           ; nincs filenév
            ;
            jsr open             ; a csatorna megnyitása
            ;
read        lda #0               ; a hibaüzenet törlése
            sta stat
            lda serial           ; a FAST bit (gyors egység) törlése
            and #%10111111
            sta serial
            ;
            ldx #15              ; 15-ös csatorna
            jsr chkout
            ;
            ldx #0               ; az olvasási parancs elküldése
            ldy #cmdlg
sendit      lda cmobuf,x
            jsr bsout
            inx
            dey
            bne sendit
            ;
```

```
          jsr clrchn            ; csatorna lezárása
          bit serial            ; ha nem gyors egység (pl. 1541)
          bvc error             ; ugrás az 'error' rutinra
          ;
          sei                   ; megszakítások letiltása
          bit dlicr
          ldx cmobuf+5          ; X = olvasandó szektorok száma
          lda d2pra             ; megszakító regiszter alapállapotba
          eor #%00010000        ;  állítása
          sta d2pra
          ;
readit    lda #8                ; várakozás a soros kapu
wait1     bit dlicr             ;  által okozott megszakításra
          beq wait1
          ;
          lda d2pra             ; CLK invertálása
          eor #%00010000
          sta d2pra
          ;
          lda dlsdr             ; az egység által küldött statusz
          sta stat              ;  byte tárolása
          and #%00001111        ; s annak ellenőrzése, nem történt-e hiba
          cmp #2
          bcs error             ; ha stat>2, akkor ugrás 'error'-ra
          ;
          ldy #0                ; byte-mutató
          lda scsize            ; lapmutató beállítása
          sta sector
toprd     lda #8                ; várakozás a soros kapu által
wait2     bit dlicr             ;  okozott megszakításra
          beq wait2
          ;
          lda d2pra             ; CLK invertálása
          eor #%00010000
          sta d2pra
          ;
          lda dlsdr             ; a soros kapuban levő adatbyte tárolása
          sta (buffer),y
          iny                   ; bytemutató csökkentése
          bne toprd             ; van még byte --> következő adatbyte olvasása
          dec sector            ; lapmutató csökkntése
          beq donsec            ; nincs több --> donsec
          inc buffer+1          ; van még --> lapmutató növelése
          jmp toprd             ; következő adatbyte olvasása
donsec    dex                   ; a szektorok számának csökkentése
          beq donerd            ; ha nincs --> vége
          ;
          inc buffer+1          ; lapmutató növelése
          jmp readit            ; következő szektor olvasása
          ;
donerd    cli                   ; megszakítások engedélyezése
          clc                   ; nem történt hiba
          lda #15
          jsr close             ; parancs csatorna lezárása
```

```
        rts                     ; vissza a hívó programhoz
        ;
error   cli                     ; megszakítások törlése
        sec                     ; C magas = hiba történt
        lda #15
        jsr close               ; parancs csatorna lezárása
        rts                     ; vissza a hívóhoz
        ;
        ; write
writsp  lda #<buffad            ; puffer kezdőcímének beállítása
        sta buffer
        lda #>buffad
        sta buffer+1
        ;
        lda #15                 ; OPEN 15,8,15
        ldx #8
        ldy #15
        jsr setlfs              ; beállítása
        ;
        lda #0
        jsr setnam              ; nincs név
        ;
        jsr open                ; megnyitás
        ;
write   lda #0                  ; hibajelző törlése
        sta stat
        lda serial              ; a gyors adatátvitelt jelző bit törlése
        and #%10111111
        sta serial
        ;
        ldx #15
        jsr chkout              ; parancs csatorna kijelölése
        ;
        ldx #0                  ; olvasási parancs elküldése
        ldy #cmdlg
send    lda cmibuf,x
        jsr bsout
        inx
        dey
        bne send
        ;
        jsr clrchn              ; eredeti csatornák visszaállítása
        ;
        bit serial              ; ha nem gyors --> 'error1'
        bvc error1
        ;
        sei                     ; megszakítások letiltása
        lda #clkin
        sta oldclk              ; órajel beállítása
        ;
        ldy #0                  ; byte mutató beállítása
        ldx cmibuf+5            ; szektorszám beállítása
writit  jsr spout               ; írás előkészítése
        lda scsize
```

```
        sta sector           ; lapmutató beállítása
        ;
topwr   lda d2pra            ; az A regiszter stabil értékének

        cmp d2pra            ; a megvárása
        bne topwr
        ;
        eor oldclk           ; a vonal invertálása
        and #clkin
        beq topwr
        ;
        lda oldclk
        eor #clkin
        sta oldclk
        ;
        lda (buffer),y
        sta dlsdr            ; az adatbyte beírása a soros kapuba
        ;
optw    lda #8               ; várakozás a soros kapú által
wait3   bit dlicr            ;  okozott megszakításra
        beq wait3
        ;
        iny                  ; byte-mutató csökkentése
        bne topwr            ; van még byte --> kiírjuk
        dec sector           ; lapmutató csökkentése
        ;
        beq donsic           ; kész --> donsic
        inc buffer+1         ; puffer mutató növelése
        jmp topwr            ; következő byte kiírása
        ;
donsic  jsr spinp            ; olvasás előkészítése
        bit dlicr            ; megszakítások inicializálása
        jsr clklo            ; CLK alacsonyra állítása
        ;
        lda #8               ; várakozás a statusz byte-ra
wait4   bit dlicr
        beq wait4
        ;
        lda dlsdr            ; statusz byte tárolása
        sta stat
        jsr clkhi
        ;
        lda stat             ; hiba ellenőrzése
        and #$0f
        cmp #$02             ; stat>2 --> 'error1'
        bcs error1
        ;
        dex                  ; szektorszám csökkentése
        beq donewr           ; nincs több szektor --> donewr
        ;
        inc buffer+1         ; puffermutató növelése
        jmp writit           ; következő szektor kiírása
        ;
donewr  cli                  ; megszakítások engedélyezése
```

```
        clc                    ; hiba nem történt jelzése
        lda #15
        jsr close              ; hibacsatorna lezárása
        rts                    ; vissza a hívóhoz
        ;
error1  cli                    ; megszakítások engedélyezése
        lda #15
        jsr close              ; hibacsatorna lezárása
        sec                    ; hiba történt jelzése
        rts                    ; vissza a hívóhoz
;
spout   lda mmureg             ; MMU-ban az írás beállítása
        ora #$08
        sta mmureg
        lda #$7f               ; soros kapu szerinti megszakítás
        sta dlicr              ;  engedélyezése
        lda #0
        sta dltimh             ; a kisiftelés idejének beállítása
        lda #$04
        sta dltiml
        lda dlcra              ; az A óraregiszter üzemmódjának beállítása
        and #$80
        ora #$55
        sta dlcra
        bit dlicr              ; a megszakítások törlése
        rts
        ;
spinp   lda dlcra              ; az A óraregiszter üzemmódjának
        and #$80               ;  visszaállítása
        ora #$08
        sta dlcra
        lda mmureg             ; az MMU-ban az olvasás jelzése
        and #$f7
        sta mmureg
        rts
        ;
clklo   lda d2pra              ; CLK alacsonyra állítása
        ora #clkout
        sta d2pra
        rts
        ;
clkhi   lda d2pra     ; CLK magasra állítása
        and #$ff-clkout
        sta d2pra
        rts
        ;
        .end
```

10. fejezet

A 80 oszlopos (VDC 8563) video chip programozása

10.1 Bevezetés

A 80 oszlopos video chipről – ha röviden is – szóltunk a 6. fejezetben a 218–223 oldalakon. Ebben a fejezetben részletesen ismertetjük használatát.

A 80 oszlopos video chipnek mindössze két regisztere található a memóriában – ezek a kommunikációs regiszterek. A két regiszter tartalma a következő:

cím	leírás	az egyes bitek jelentése							
		7	6	5	4	3	2	1	0
$D600	regiszterszám írás	--	--	R5	R4	R3	R2	R1	R0
	állapot olvasás	STATUS	LP	VBLANK	--	--	VER0	VER1	VER2
$D601	adat írás/olvasás	D7	D6	D5	D4	D3	D2	D1	D0

A VDC 8563 chip regiszterei és a video memória ezeken a kommunikációs regisztereken keresztül érhetők el, ahogy azt a 228. oldalon már leírtuk. A VER0,VER1,VER2 bitek a video chip verziószámát adják. Mint majd látjuk, bizonyos esetekben erre tekintettel kell lenni.

A VDC 8563-t elsősorban karakteres megjelenítésre tervezték. Két, egyenként 256 elemű karakterkészletet képes kezelni. Szemben azonban a VIC-vel ezeket egyszerre jeleníti meg. Rendelkezik grafikus lehetőségekkel is, ezeket azonban nem lehet a VIC fejlett grafikus lehetőségeivel összehasonlítani. Elég, ha csak a sprite-okra gondolunk! Éppen az önálló videó memória miatt a chip 2MHz-es órajellel is képes üzemelni.

10.2 A 8563 chip regiszterei

A Commodore információs kártya 36. oldala tartalmaz egy áttekintő táblázatot a chip regisztereiről, az alábbiakban részletesen felsoroljuk valamennyit. Először a regiszter elnevezését, majd az egyes bitek jelölését adjuk meg. Ezt követi a bitek használatának részletes leírása. A megadott alapértelmezést a Commodore 128-as gép C128-as módjában használja a rendszer, a CP/M mód ettől eltérő értékeket használ.

0. regiszter (vízszintes totál): HT7 HT6 HT5 HT4 HT3 HT2 HT1 HT0
A 8 bites érték két vízszintes szinkronizációs jel közti távolság – 1, karakterekben mérve. Ez magában foglalja a kijelzett karaktereket, a keret méretét és a két jelzés közti 'üresjáratot' is. A C-128 BASIC ezt 126-ra állítja be.

1. regiszter (karakter/sor): HD7 HD6 HD5 HD4 HD3 HD2 HD1 HD0
A 8 bites érték a kijelzett karakterek számát adja meg. Kezdőértéke 80.

2. regiszter (szinkron pozíció): HS7 HS6 HS5 HS4 HS3 HS2 HS1 HS0
A regiszter a vízszintes szinkronizáció helyét adja meg karakterekben. Az érték az első kijelzett karaktertől mérendő, így az értéknek az R1 regiszterben levőnél nagyobbnak kell lennie. Ez az érték határozza meg, hogy az RGBI monitoron hol jelenjen meg a kép. Kezdőértéke: 102.

3. regiszter (szinkron szélesség): SW7 SW6 SW5 SW4 SW3 SW2 SW1 SW0
Az SW3-SW0 bitek a vízszintes, az SW7-SW4 bitek a függőleges szinkron nagyságát adják meg. A vízszintes szinkron karakterben, a függőleges raszter sorban értendő. Kezdőérték: 73 (=4*16+9). Ez 9 karakternek és 4 rasztersornak felel meg.

4. regiszter (függőleges totál): VT7 VT6 VT5 VT4 VT3 VT2 VT1 VT0
Hasonlóan a 0. regiszterhez a 8 bites szám két függőleges szinkronjel közti távolság – 1, karakterekben mérve. Ez az érték magában foglalja a kijelzett sorokat, a keretet és a két jel közti 'üresjáratot'. A C128 ezt az értéket 39-re állítja be.

5. regiszter (függőleges eltolás): -- -- -- VA4 VA3 VA2 VA1 VA0
Az 5 bites szám a függőleges szinkron jelét növeli meg rasztersorban mérve. A függőleges szinkron finom hangolására való. Ha interlace módban dolgozunk, akkor az értékét a kívánt kétszeresére kell beállítani. Ezért 5 bites a regiszter. Kezdeti értéke 0. Ha a regisztert olvassuk, a felső bitek mindig 1-et adnak, s ezért a 0 kezdeti érték 224-nek látszik.

6. regiszter (sorszám/képernyő): VD7 VD6 VD5 VD4 VD3 VD2 VD1 VD0
A kijelzett sorok száma. A C-128 ezt 25-re állítja be. Ennek az értéknek kisebbnek kell lennie a 4. regiszter értékénél.

7. regiszter (függőleges szinkron): VS7 VS6 VS5 VS4 VS3 VS2 VS1 VS0
A függőleges szinkron kezdete – 1, karakterekben mérve. Az érték az első kijelzett karaktersortól számít, így nagyobbnak kell lennie, mint a 6. regiszterben levő érték. Kezdőértéke 32.

8. regiszter (interlace mód): -- -- -- -- -- -- IL1 IL0
A regiszter az ún. interlace módot állítja be. A chip összesen 3 féle raszter megjelenítési módot ismer. Ezek a következők: nincs interlace, van interlace és van interlace és video szinkronizálás is.

Ha nincs interlace szinkronizálás, akkor minden egyes raszter sor a függőleges szinkron ütemében kerül frissítésre. Ez a 00 és 10 bitkombinációknak felel meg.

Ha csak interlace szinkronizálást engedünk meg, akkor a páros és páratlan raszter sorok felváltva lesznek frissítve. Mind a függőleges, mind a vízszintes raszter sorokban ugyanaz jelenik meg, de ezek egymáshoz képest fél raszter sorral eltolva jelennek meg. Ennek következtében a karakter sokkal tömörebb lesz függőleges irányban. A hatás elsősorban kimondottan az erre tervezett monitorokon érzékelhető. Ennek a módnak a 01 bitkombináció felel meg.

Az interlace és video szinkronizálás esetén a páros és páratlan raszter sorok nem ugyanazt az információt generálják, s ennek következtében a függőleges felbontás duplájára nő. Ehhez a 8563 további regisztereit is át kell programozni. Egyes értékek (függőleges eltolás, függőleges szinkron stb.) csak párosak lehetnek.

A C-128 nem használ interlace szinkronizálást, ezért IL1 és IL0 értéke 0. Ha a regisztert kiolvassuk, akkor 252-t kapunk, mert a nem használt bitek 1-et adnak.
9. regiszter (karakterméret): -- -- -- CV4 CV3 CV2 CV1 CV0
A karaktert alkotó rasztersorok száma – 1. A méretbe a karaktersorok közti rés nagyságát is be kell számítani. Kezdeti értékét a rendszer 7-nek állítja be, ez 8 rasztersornak felel meg. Ez kiolvasáskor 231-et ad.

10. regiszter (kurzor mód): -- CM1 CM0 CS4 CS3 CS2 CS1 CS0
A CM1 és CM0 bitek a kurzor megjelenésének módját határozzák meg. Az egyes értékek jelentése a következő:

CM1	CM0	kurzor típusa
0	0	álló kurzor
0	1	nincs kurzor
1	0	lassú villogás
1	1	gyorsabb villogás

A CS4-CS0 bitek a kurzor első sorát adják meg. Ettől a rasztersortól kezdve villog a kurzor karaktere.

11. regiszter (kurzor típusa): -- -- -- CT4 CT3 CT2 CT1 CT0
A regiszter az utolsó rasztersort adja meg a karakteren belül, ahol a kurzor még villog. A CS és CT bitek együtt határozzák meg, hogy a kurzor helyén álló karakter mekkora darabja villog.

12. regiszter (képernyő memória HI): VM15 VM14 VM13 VM12 VM11 VM10 VM9 VM8
13. regiszter (képernyő memória LO): VM7 VM6 VM5 VM4 VM3 VM2 VM1 VM0
A két regiszter együttesen meghatározza, hogy a video memóriában hol kezdődik az a rész, ami alapján a rendszer a képernyőn levő információt generálja. A képernyő kódok sorfolytonosan helyezkednek el, úgy mint a VICII chip esetében. Eltérően a processzor címektől, először a 16 bites cím felső, azután az alsó byte-ját tárolja a chip. A C-128 a képernyő memóriát a $0000 címre helyezi (a képernyő memóriában).

Grafikus üzemmódban ez a cím a bit térkép elejére mutat és az első nyolc képpont adatait tartalmazza. (Magas bit= bekapcsolt pont) Eltérően a VICII-től ebben az esetben is **sorfolytonos a tárolás**: az egyes képpontokat rasztersoronként kell elhelyezni!

A 80*25-ös karakteres képernyőhöz 80*25=2000 byte-ra, míg a 640*200 képpontból álló grafika megjelenítéséhez 640*200/8=16000 byte-ra van szükség.

14. regiszter (kurzor helye HI): CP15 CP14 CP13 CP12 CP11 CP10 CP9 CP8
15. regiszter (kurzor helye LO): CP7 CP6 CP5 CP4 CP3 CP2 CP1 CP0
A két regiszter együtt a video memória címét határozza meg. Ha ennek a címnek a képernyőn egy látható karakter felel meg, akkor a 10-11. regiszterek tartalmának megfelelő módon ezen a helyen a kurzor villogni fog. Ha a kurzor módját 'nincs kurzor'-nak választottuk, akkor a kurzor termeszetesen nem villog.

16. regiszter (fényceruza vízszintes): LH7 LH6 LH5 LH4 LH3 LH2 LH1 LH0
17. regiszter (fényceruza függőleges): LV7 LV6 LV5 LV4 LV3 LV2 LV1 LV0
A két regiszter együtt a fényceruza pozícióját adja. Azt, hogy ez valódi érték-e, a kommunikációs regiszter 6., LP bitje mondja meg. Ha a bit alacsony(0), akkor a regiszterek nem tartalmaznak értékes adatot. Ha igen, akkor a regiszterekbe a fényceruza valódi adatai íródtak be. A 16. vagy 17. regiszter kiolvasása az LP bitet törli. A függőleges érték 1-től, míg a vízszintes érték 8-tól kezdődik.

18. regiszter (aktuális byte HI): UA15 UA14 UA13 UA12 UA11 UA10 UA9 UA8
19. regiszter (aktuális byte LO): UA7 UA6 UA5 UA4 UA3 UA2 UA1 UA0
A két regiszter együtt egy 16 bites címet határoz meg a chip saját video memóriájában. A 18. és 19. regiszterek bármelyikébe való írás esetén a 8563-as video chip automatikusan beolvassa a megcímzett byte-ot a video memóriából a 31. regiszterbe. Az olvasás sikeres befejezését a kommunikációs regiszter STATUS bitje jelzi: magas(1) lesz. Ekkor lesz a 31. regiszterben értékes adat. Ezzel egyidőben a VDC 8563 eggyel megnöveli a 18-19. regiszterekben levő címet. Az adatok folyamatos olvasásához tehát nem kell a címet programból növelni.

Az írás hasonlóan történik. Ha a 31. regiszterbe írunk egy adatot, akkor a 18-19. regiszterek által meghatározott címre másolja a byte-ot a chip, majd megnöveli a címet és az annak megfelelő adatot beolvassa a 31. regiszterbe. A művelet befejezésekor a STATUS bit magas(1) lesz.

20. regiszter (szín memória HI): VC15 VC14 VC13 VC12 VC11 VC10 VC9 VC8
21. regiszter (szín memória LO): VC7 VC6 VC5 VC4 VC3 VC2 VC1 VC0
A két regiszter együtt egy 16 bites címet határoz meg a video memóriában. Itt kezdődik a szín vagy attributum memória. Az attributum memória byte-jainak jelentéséről a 222. oldalon szóltunk. A színmemória kezdőértéke $800.

22. regiszter (mátrix vízszintes, függőleges): MH3 MH2 MH1 MH0 MV3 MV2 MV1 MV0
Az MV3-MV0 bitek a karakter szélességét adják meg. Ebbe a kijelzett karakter szélessége és a karakterek közti üres rész is beleértendő. Ha ilyen rész nincs, akkor ez az érték egyenlő az MH3-MH0 értékkel + 1. Ha van karakterek közti rész, akkor az MV3-MV0 bitek a karakter kijelzett része - 1 értéket adják pontokban mérve. Ha dupla széles karaktereket használunk, akkor a fenti értékhez hozzá kell még 1-et adni.

Az MH3-MH0 bitek a karakter vizszintes hossza - 1 értéket tartalmazzák pontokban mérve. Ez az érték a kijelzett rész és a karakterek közti üres részt is tartalmazza. Ha dupla széles kijelzést használunk, akkor az értékből az 1-et nem kell levonni. A 22. regiszter kezdeti értékét a C-128 120(=7*16+8)-ra állítja be. Ennek megfelelően egy karakter 8 raszterpont széles és nincs a karakterek közti üres hely.

23. regiszter (függőleges kijelzés): -- -- -- CD4 CD3 CD2 CD1 CD0
A CD4-CD0 bitek eggyel kisebb értéket adnak, mint ahány raszter sor van egy karakterben. Összehasonlítva ezt a 9. regiszter CV4-CV0 értékeivel megkapjuk a sorok közti üres rasztersorok számát. A CD4-CD0 érték kisebb kell hogy legyen a CV4-CV0 értéknél. Kezdeti értéke 7, amit 232-nek lehet kiolvasni, mert a nem használt felső négy bit 1-et ad. Ez az érték 8 rasztersornak felel meg.

24. regiszter (függőleges eltolás): COB RVS CRT SV4 SV3 SV2 SV1 SV0
Az SV4-SV0 bitek a függőleges eltolást adják meg raszter sorokban. A képernyő görgetéséhez lehet használni. Ha pl. értéke 1, akkor az első kijelzett raszter sor az első karaktersor második rasztersora lesz. Az utolsó sorban egy új karaktersor első rasztersora jelenik meg. Kezdőértékét a C-128 8-ra állítja be.

A CRT bit a villogó karakterek villogásának gyorsaságát szabályozza. Ha ez a bit alacsony(0), akkor a karakterek lassabban, ha magas(1), akkor gyorsabban villognak.

A RVS bit magasra(1) állításával az előtér/háttér színe megcserélődik.

A COB bit a video memória blokk másolásához tartozik. Használatát a 30. regiszternél ismertetjük.

25. regiszter (vízszintes eltolás): TXT ATR SEM DBL SH3 SH2 SH1 SH0
Az SH3-SH0 bitek a vízszintes eltolás mértékét határozzák meg. A képernyő finom görgetéséhez lehet használni. A rendszer ezt 0-ra állítja be.

A DBL bit a 80 oszlopos képernyőből 40 oszloposat készít. Ha a bit magas, minden egyes képernyőpont szélessége kétszeresére nő. Természetesen a vízszintes megjelenítést befolyásoló értékeket újra kell programozni.

A SEM bit lehetővé teszi kvázigrafikus karakterek kiíratását. A karakter kijelzett szélessége maximum 8 raszterpont lehet. Ha a SEM bitet magasra állítjuk a két karakter közti üres részen (ami normálisan háttér színű) a chip automatikusan megismétli az utolsó pont oszlopát.

Az ATR bit magas(1) volta jelzi, hogy a chip használja-e az attributum memóriát vagy sem. Több esetben nincs szükség erre, ilyenkor a video memóriának ez a része másra használható. Ezt az ATR alacsonyra(0) állításával érhetjük el. Ekkor minden karakter a 26. memóriában levő előtér színnel jelenik meg. Mindössze egyetlen 256 karakterből álló karakterkészlet használható.

A TXT bittel kapcsolhatunk át karakteres üzemmódból grafikusra és vissza. Karakteres kijelzés esetén ez a bit 0, különben 1.

A C-128 ennek a regiszternek a tartalmát 64-re állítja be, ami az attributum bekapcsolását jelenti.

26. regiszter (előtér/háttér színe): FG3 FG2 FG1 FG0 BG3 BG2 BG1 BG0
Az FG3-FG0 és BG3-BG0 bitek az előtér, illetve a háttér színét határozzák meg. A háttér szín a keret színe egyben. Az előtér színét a chip csak az attributumok kikapcsolása esetén (ATR=0) használja. A regiszter kezdeti értéke 240.

27. regiszter (címnövelés): AI7 AI6 AI5 AI4 AI3 AI2 AI1 AI0
Lehetőség van arra, hogy minden egyes karakter sor kijelzése után adott byte-ot a video memóriában átugorjon a 8563-as chip. Ennek hatására a kijelzett részt nem csak raszter, hanem karakter soronként is el lehet tolni. A soronként kihagyott byte-ok számát ez a regiszter adja. A C-128 ennek értékét 0-ra állítja be.

28. regiszter (karaktermemória HI): CM2 CM1 CM0 RAT -- -- -- --
A regiszter CM2-CM0 bitjei a karaktergenerátor helyét határozzák meg a video memóriában. Ezek a bitek a kezdőcím felső három bitjét adják meg. Egy karakter memória rész 8192 byte-ból áll. A C-128 a karaktergenerátort $2000-re helyezi.
A RAT bit a használt video memória típusát adja meg. Mivel a hardver adott, értéke kötelezően 0. A regiszter tartalma tehát 47. A nem használt bitek 1-esnek olvashatók ki.

29. regiszter (aláhúzott sor): -- -- -- UL4 UL3 UL2 UL1 UL0
Ennek a regiszternek a tartalma adja meg, hogy hányadik raszter sort használja a rendszer aláhúzásra. A 0 érték a karakter első raszter sorának felel meg. Ha pl. 3-ra állítjuk, akkor a betűk nem alá-, hanem áthúzva jelennek meg. Kezdeti értéke: 7. Ez kiolvasva 231.
30. regiszter (szószámláló): WC7 WC6 WC5 WC4 WC3 WC2 WC1 WC0
A 8563 chip képes a video memória másolására, illetve egyetlen karakterrel való feltöltésére. Egyetlen megkötés, hogy ezt legfeljebb csak 256 byte-tal lehet megtenni.

A karakter feltöltés hasonló a video memóriába való írással. A 18–19. regiszterekbe betöltjük az első írandó byte címét, majd a 31. regiszterbe beírjuk a feltöltendő byte-ot. Ezt követően a 24. regiszter COB bitjét alacsonyra állítjuk, s a 30. regiszterbe beírjuk a feltöltendő byte-ok számát. A regiszterbe való írás indítja el a feltöltést. A chip az adott számú byte-ba beírja a 31. regiszterben levő byte-ot. A művelet végén a 18–19. regiszterek az első olyan címre mutatnak, ahová nem került a karakter bemásolásra. Mivel a 31. regiszterbe való írás már maga beír egy byte-ot a video memóriába, ezért a 30. regiszterbe már eggyel kisebb számot kell írni, mint ahány helyet a karakterrel fel akarunk tölteni.

A video memória másolásakor ugyancsak a 30. regiszter tartalmazza az átmásolandó byte-ok számát.

31. regiszter (adatregiszter): DA7 DA6 DA5 DA4 DA3 DA2 DA1 DA0
Az adatregiszter tartalmazza a 18–19. regiszterek által megcimzett video memóriában levő byte-ot.

32. regiszter (blokk kezdete HI): BA15 BA14 BA13 BA12 BA11 BA10 BA9 BA8
33. regiszter (blokk kezdete LO): BA7 BA6 BA5 BA4 BA3 BA2 BA1 BA0
A 32–33. regiszterek egy 16 bites címet határoznak meg. A blokk másolás alkalmazásakor a video memória 32–33. regiszterei által meghatározott címtől a 30. regiszterben levő byte-ot másol át a chip a 18–19. regiszterek által meghatározott címre. A másolás (hasonlóan a video memória feltöltéséhez) a 30. regiszterbe történő íráskor kezdődik meg. Előtte kell beállítani a két címet, illetve a COB bitet. Ez utóbbi értéke 1 kell, hogy legyen. (Ellenkező esetben nem másolás, hanem feltöltés történik).

34. regiszter (DISPEN kezdete): DS7 DS6 DS5 DS4 DS3 DS2 DS1 DS0
35. regiszter (DISPEN vége): DE7 DE6 DE5 DE4 DE3 DE2 DE1 DE0
A függőleges és vízszintes szinkronizáció alatt az RGBI információknak alacsonynak kell lenni (elkerülendő a képernyő villódzását). A két regiszter megadja, hogy mettől-meddig kell az RGBI információt törölni. Erre az időre az RGBI lábakon alacsony feszültség (=0V) jelenik meg. A C-128 ezt a két értéket 125-re és 64-re állítja be.

36. regiszter (DRAM frissítés): -- -- -- -- -- RR3 RR2 RR1 RR0
Az RR3-RR0 értékek megadják, hogy egy raszter sorra hány memória frissítés jusson. Ez a frissítés mind a kijelzett, mind az üres sorokra vonatkozik. A C-128 ezt az értéket 5-re állítja be. Kiolvasáskor ez 245-t ad, mert az értéktelen biteket a chip 1-nek olvassa.

10.3 A 8563 chip hardver specifikációja

Kapcsolat a processzorral

A 8563 és a 8502 (vagy Z80) közötti kapcsolat a lehető legegyszerűbb (hála a kommunikációs regisztereknek). A kapcsolat az alábbi jelek segítségével valósul meg:

- D_0–D_7: Kétirányú adatbusz, amelyiken a 8502 és a 8563 adatot cserél.
- CS: chip kiválasztó jel. Ez a jel magas kell hogy legyen a chip helyes működéséhez. A jelet a PLA állítja elő.
- /CS: chip kiválasztó jel. Ennek alacsonynak kell lennie a chip helyes működéséhez. Ugyancsak a PLA állítja elő.
- /RS: regiszter kiválasztása. Ha a vonal magas, akkor lehetőség van a regiszterek olvasására és írására. Ha alacsony, akkor lehetőség van az állapotregiszter olvasására és az adatregiszter írására.

- R/W: az adatbusz irányát szabályozó jel.

- INIT: alacsonyra vitele törli a chipet. A C–128-on ez a vonal össze van kötve a /RES vonallal.
- DISPEN: a kijelzést engedélyező output jel. A C–128 nem használja.
- RES: Ez a jel inicializálja a video kijelzést irányító számlálókat. A regiszterek tartalma nem változik. A C–128 nem használja, be sincs kötve.
- TST: tesztelési célokra szolgál. A C–128-on le van földelve.

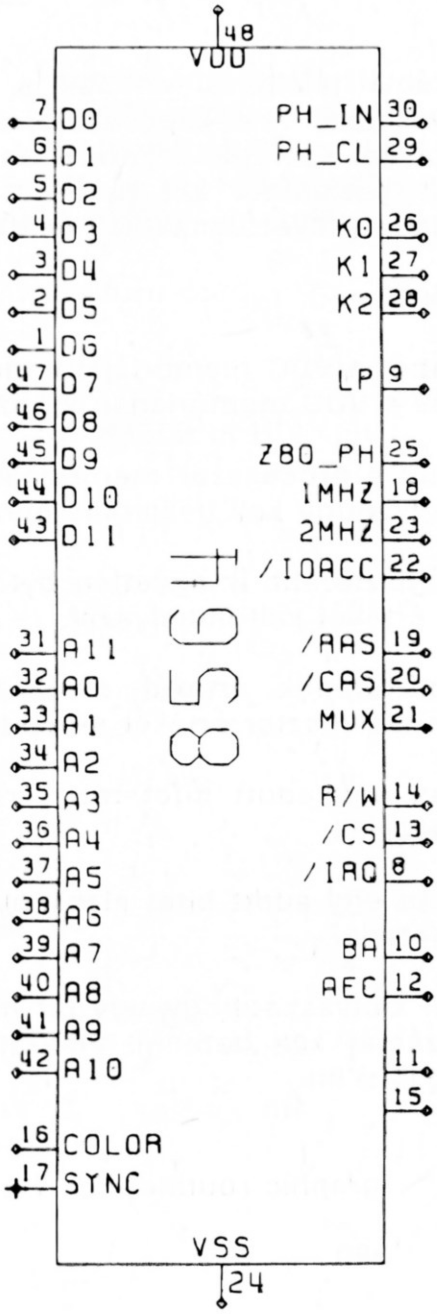
A következő legfontosabb jelcsoport az, amivel a 8563 a saját video memóriájával tart kapcsolatot. Ezek a következők:

- DD_0–DD_7: Kétirányú lokális adatbusz a DRAM-hoz.
- DA_0–DA_7: multiplexelt címbusz.
- DR/W: a lokális busz irányát meghatározó jelző.
- /RAS: sor cím jelző a DRAM-nak.
- /CAS: oszlop cím jelző a DRAM-nak.

Az utolsó csoportba azok a jelek tartoznak amelyek az RGBI monitor képét előállítják:

- DCLK: video pont órajel. Meghatározza a pont méretét. Ezt használja a chip valamennyi belső időzítéséhez.
- CCLK: karakter órajel, a C–128 nem használja.
- LP2: bemenet a fényceruzától. Ezen a lábon egy pozitiv átmenet beírja a fényceruza helyzetét a megfelelő regiszterekbe.
- HSYNC: szinkronizációs jel.
- R,G,B,I: kimenő szín és intenzitás jelek.

A 8563 chip lábkiosztása



10.4 Programozási példák

Az alábbiakban egy rutingyűjteményt adunk meg, amelyek a 80 oszlopos video chip grafikájának használatát könnyítik meg: CP/M-ben. A könyv végén példát mutatunk olyan Turbo Pascal példákra, melyek EXTERNAL eljárások és függvényekként használják ezeket.

A program elején egy ugrótábla található, aminek a segítségével lehet az egyes rutinokat használni. A Turbo Pascallal való kapcsolattartás miatt a rutinok a veremből kapják a paramétereiket. A leírásnál szereplő adatokat az ott megadott sorrendben kell a verembe helyezni. Mindegyik paraméter két byte-os. Ha értelemszerűen csak egy byte-os érték lehet, akkor a felső byte-ot 0-nak kell beállítani.

A rutinok funkciója rendre a következő:

move$to A proceszor memóriából a VDC memóriájába másol. A verembe a kezdőcımet, a másolandó byte-ok számát és a VDC memóriában a kezdőcímet kell beállítani.

move$from A VDC memóriájából a processzor memóriájába másol. A verembe az adatokat a rutin meghívása előtt ugyanúgy kell beállítani, mint az előző esetben.

VDCregwrite A VDC adott regiszterébe ír egyetlen byte-ot. A veremben az adatokat, a regiszter sorszámát, s annak értékét kell behelyezni.

VDC$read A VDC adott regiszterének értékét olvassa ki. A veremre a regiszter sorszámát kell helyezni. A rutin a regiszter értékét visszateszi a veremre.

set$point A VDC memóriájában egy adott bitet magasra állít. A verembe a memória címét és a bit sorszámát kell betenni.

reset$point A VDC memóriájában egy adott bitet alacsonyra állít. A verembe a memória címét és a bit sorszámát kell betenni.

test$point A VDC memóriájából kiolvassa, hogy egy bit magas, vagy sem. A verembe a memória címét és a bit sorszámát kell betenni. Visszatéréskor a verem tetején levő érték 0, ha alacsony; 1, ha magas a bit.

```
                title       'graphic routines for Turbo'

                maclib      z80


                org         0e000h
io$1            equ         0ff04h
bank$1          equ         0ff02h

;                           jump table for routines
;
;

        jp      move$to     ; from memory to video
        jp      move$from   ; from video to memory
```

```
        jp      VDC$reg$write   ; VDC register write routine
        jp      VDC$reg$read    ; VDC register read routine
        jp      set$point       ; turn a pont on
        jp      reset$point     ; turn a point off
        jp      test$point      ; test if a point on


move$to:
        pop     h               ; return address in HL
        pop     d               ; FROM in DE
        pop     b               : NUMBER in BC
        xthl                    ; TO in HL, return address on stack
        push    b               ; NUMBER on stack

sta     io$1                    ; io turn on!

move$char$to:
        call    set$update$adr
        pop     hl              ; NUMBER in HL

move$one$char$to:
        ldax    d
        outp    a
        dcr     c
        inx     d

write$wait:
        inp     a
        ral
        jrnc    write$wait

        inr     c
        dcx     h
        mov     a,h
        ora     l
        jrnz    move$one$char$to

        sta     bank$1                  ; io turn off
        ret
;
;
;
wait:
        push    psw
        lxi     b,0d600h
wait$loop:
        inp     a
        ral
        jrnc    wait$loop
        pop     psw
        outp    a
        inr     c
        ret
;
```

```
;
;
set$update$adr:
        mvi     a,18
        call    wait
        outp    h
        mvi     a,19
        call    wait
        outp    l
        mvi     a,31
        call    wait
        dcr     c

update$wait:
        inp     a
        ral
        jrnc    update$wait
        inr     c
        ret

;
;                               VDC register write
;

VDC$reg$write:
        pop     h
        pop     de
        xthl
        mov     a,l
        call    wait
        outp    e
        ret

;
;                               VDC register read
;

VDC$reg$read:
        pop     hl
        xthl
        mov     a,l
        call    wait
        inp     l
        xthl
        push    h
        ret

;
```

```
;           set$point
;
;           hl = pointer to video memory
;           a  = bit affected
;

set$point:
            pop     h                       ; return address
            pop     d                       ; de = bit affected
            xthl

            push    h
            call    set$update$adr
            inp     b
            mov     a,e
            slar    a
            slar    a
            slar    a
            ori     0c7h                    ; = %11000111
            sta     set$bit+1

            mov     a,b
set$bit:
            setb    a,0
            pop     h
            push    psw
            call    set$update$adr
            pop     psw
            outp    a
            ret

;
;           reset point
;
;           hl = pointer to video memory
;           a  = bit affected
;

reset$point:
            pop     h                       ; return address
            pop     d                       ; de = bit affected
            xthl

            push    h
            call    set$update$adr
            inp     b
            mov     a,e
            slar    a
            slar    a
            slar    a
            ori     87h                     ; = %10000111
            sta     reset$bit+1

            mov     a,b
```

```
reset$bit:
        setb    a,0
        pop     h
        push    psw
        call    set$update$adr
        pop     psw
        outp    a
        ret

;
;       test if point is on
;
;       hl = pointer to video memory
;       a  = bit affected
;

test$point:
        pop     h               ; return address
        pop     d               ; de = bit affected
        xthl                    ; a  = bit affected

        call    set$update$adr
        inp     b
        mov     a,e
        slar    a
        slar    a
        slar    a
        ori     47h             ; = %01000111
        sta     test$bit+1

        mov     a,e
test$bit:
        bit     a,0
        jrnz    no$point
        lxi     h,1
        xthl
        push    h
        ret

no$point:
        lxi     h,0
        xthl
        push    h
        ret

;
;                       move from video to memory
;
```

```
move$from:
        pop     h
        pop     d
        pop     b
        xthl
        push    b
        sta     io$1

        call    set$update$adr
        pop     h

move$from$one:
        dcr     c

wait$from:
        inp     a
        ral
        jrnc    wait$from
        inr     c
        inp     a
        stax    d

        inx     d

        dcx     h
        mov     a,h
        ora     l
        jrnz    move$from$one

        sta     bank$1
        ret

        end
        ret

        end
```

Arra az esetre, ha valaki nem rendelkeznék makro assemblerrel megadjuk a fenti file fordítás után keletkezett .HEX típusú file tartalmát. Ez ASCII file, így pl. az ED-vel vagy a Turbo Pascal szövegszerkesztőjével is beírhatjuk.

```
:10E00000F215E0F2DAE0F25EE0F268E0F273E0F2DC
:10E0100094E0F2B5E0E1D1C1E3C53204FFCD43E0C5
:10E02000E11AED790D13ED781730FB0C2B7CB52040
:10E03000F03202FFC9F50100D6ED781730FBF1EDA3
:10E04000790CC93E12CD35E0ED613E13CD35E0EDE2
:10E05000693E1FCD35E00DED781730FB0CC9E1D1DD
:10E06000E37DCD35E0ED59C9E1E37DCD35E0ED68E7
:10E07000E3E5C9E1D1E3E5CD43E0ED407BCB27CB40
:10E0800027CB27F6C7328AE078CBF8E1F5CD43E01D
:10E09000F1ED79C9E1D1E3E5CD43E0ED407BCB275C
:10E0A000CB27CB27F68732ABE078CBF8E1F5CD4331
:10E0B000E0F1ED79C9E1D1E3CD43E0ED407BCB2741
:10E0C000CB27CB27F64732CBE07BCB78200621014C
:10E0D00000E3E5C9210000E3E5C9E1D1C1E3C532B0
:10E0E00004FFCD43E0E10DED781730FB0CED781225
:0AE0F000132B7CB520F03202FFC9AB
:0000000000
```

11. fejezet

A CP/M+ V3.0 használata

11.1 A lemezegységek használata

A 4. fejezetben a Commodore 128 CP/M+ operációs rendszerének alapjait ismertettük. A mostani fejezet az ott közöltektől lényegesen mélyebben tárgyalja az operációs rendszer egyes funkcióit, illetve azok használatát. A fejezet megértéséhez ismerni kell a 4. fejezetben elmondottakat, továbbá rendelkezni kell bizonyos Z80-as alapismeretekkel. Ez utóbbi az ajánlott irodalomjegyzék Z80-as könyveiből megszerezhető.

Több lemezegység használata
A C-128 CP/M+ (szemben a C-64 CP/M bővítésével) lehetőséget biztosít több lemezegység használatára is. Ebben az esetben az operációs rendszer betöltésére nem használhatjuk az 'autoboot' funkciót.

Először a demonstrációs lemezen levő program segítségével a második, harmadik lemezegységet átszámozzuk a 9-es, 10-es hardver számú egységekre. Ezután - nem kikapcsolva a C-128-at - behelyezzük a 8-as lemezegységbe a CP/M+ rendszerlemezt és kiadjuk a BOOT BASIC parancsot. Ennek hatására a C-128 betölti és elindítja a CP/M+ operációs rendszert. A BOOT nem inicializálja a lemezegységeket, ezért a rendszer bejelentkezése után is 9-es, 10-es egységként használhatjuk az átszámozott egységeket. A CP/M-ből ezekre A:, B:,C: névvel használhatjuk.

Ha a lemezegységeket **fizikailag is átszámoztuk**, akkor a szokásos eljárást használhatjuk: a rendszerlemezt a gép bekapcsolása előtt behelyezzük a 8-as lemezegységbe, majd bekapcsoljuk a gépet.

A CP/M+ rendszer használatához célszerű két gyors (1570 vagy 1571 jelű) lemezegységet használni. Különösen a másolások gyorsulnak meg; pl. a COPY b:*.*=a: parancs hatására a teljes a: lemez átmásolódik a b: lemezre. Ha egyetlen lemezegységünk van csak, akkor a fiktiv e: lemezegységet kell használnunk, s file-onként cserélni a lemezeket!

Fenntartott file-név típusok
A CP/M+ rendszerben a file-ok nevei egy maximum három karakterből álló típussal (kiterjesztéssel) végződhetnek. Ezek közül bizonyosakat a rendszer parancsai és a nyelvi fordítók használnak, ezeket nem célszerű másra használni. Ezek - a rendszer jelenlegi fejlesztési stádiumában - a következők:

File típus	Jelentés
ASM	assembler forrás nyelvi lista
BAS	BASIC forrás nyelvi lista
COM	Z80 abszolút című gépi program
HEX	Intel HEX formájú program
HLP	információs file-ok
$$$	ideiglenes file-ok
PRN	MAC és RMAC nyomtatási file-ja
REL	RMAC outputja (LINK-hez kell)
SUB	SUBMIT-hoz adatfile
SYM	szimbolum tábla (MAC, RMAC vagy LINK)
RSX	rezidens rendszerbővítő file

Lemezformátumok

A C-128 CP/M+ több fajtájú lemez írását/olvasását, egyesek formázását támogatja. A rendszernek ez a része még jelenleg is fejlesztés alatt áll, ezért további módosítások várhatók.

A C-128 CP/M+ három Commodore GCR formátumú lemezt támogat. Ezek sorban a következők:

C-64 egy oldalas lemez: ez a lemezformátum azonos a CP/M 2.2 által a Commodore 64-en használt lemezformátummal. Az FCB ebben az esetben 32 sávot, sávonként 17 szektort tartalmaz. A BIOS automatikusan 1-gyel megnöveli a sáv számot, ha az 18, vagy annál nagyobb. (Az I. kötetben szereplő, a lemezre közvetlenül író/olvasó példa ugyanezt a módszert használja!) A rendszer 0. és 1. sávja (fizikailag az 1. és a 2.) fenntartott a rendszerfile-ok számára.

C-128 egy, illetve kétoldalas lemez: A második lemezformátumot már a 1570 és 1571-es lemezegységekre tervezték, s a lemez valamennyi blokkját használja. Ezt azzal a fogással érték el, hogy az FCB 638 (1276) 1 szektoros sávnak tekinti a lemezt. A logikai és a fizikai sávok közti transzformációt a következő táblázat adja meg:

Logikai sáv(S)	Fizikai sáv
000-354	((S+2)/21)+1
355-486	((S-354)/19)+18
487-594	((S-487)/18)+25
595-638	((S-595)/17)+31
639-1276	a második oldalon S-639-el s a fenti átszámítással

A fenti képletben az osztásnak természetesen az egész részét kell venni. A maradék a logikai szektor számot adja, amit még az adatátvitel gyorsítása érdekében tovább transzformál a rendszer:

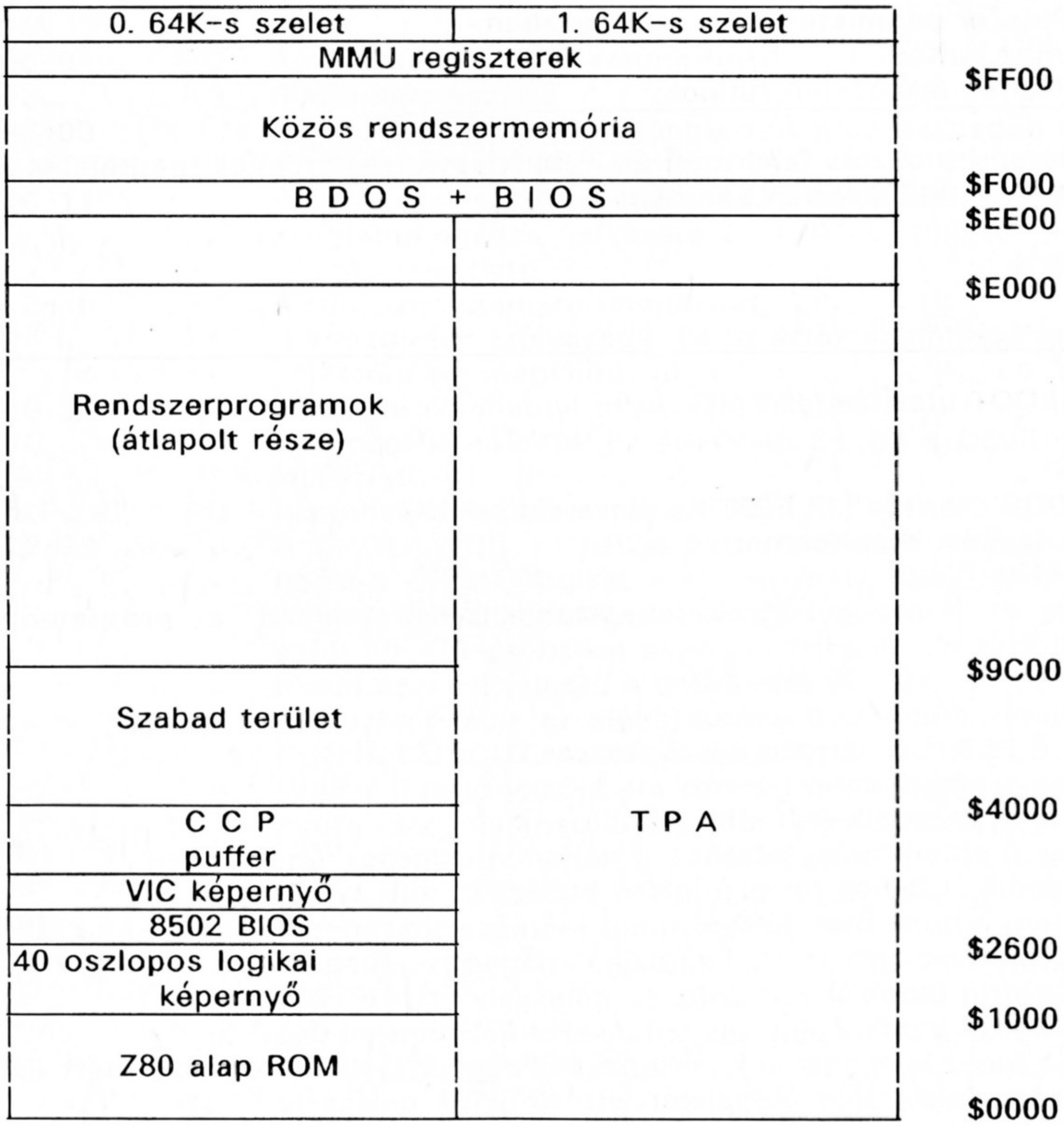
Sávok	Szektorok																				
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0-16	00	17	13	09	05	01	18	14	10	06	02	19	15	11	07	03	20	16	12	08	04
17-23	00	04	08	12	16	01	05	09	13	17	02	06	10	14	18	03	07	11	15		
24-29	00	11	04	15	08	01	12	05	16	09	02	13	06	17	10	03	14	07			
30-34	00	07	14	04	11	01	08	15	05	12	02	09	16	06	13	03	10				

11.2 Memóriaszervezés

Már a 4. fejezetben is utaltunk rá, hogy a CP/M+ a memóriát a C–128–as módtól eltérően használja. A 4. fejezetben levő általános sémával ez ugyan összhangban van, de a CCP, a BIOS és a BDOS rendszerkomponenseknek a TPA területen csak darabjai vannak, aminek segítségével a CP/M+ képes ezeknek a rutinoknak a másik 64K–s memória szeleten levő darabjait meghívni. Ez még egy 4 Kbyte–os ROM–ot is tartalmaz.

A TPA terület 64K–s szelete mindössze annyi információt és programrészt tartalmaz, ami elegendő arra, hogy a BDOS, illetve BIOS rutinok az átlapolt részre adják a vezérlést. Magán az átlapolt részen találhatók a rutinok 'érdemi' részei. Ez a technika eredményezi, hogy a felhasználói programok számára 59 Kbyte áll rendelkezésre, ami 8 bites gép esetén igen nagy terület.

CP/M+ memóriafelhasználás



A 0. szelet használata

Cím	Leírás
0000 - 0FFF	Z80-as ROM (4Kbyte)
1000 - 13FF	billentyűzet definíció
1400 - 23FF	80 oszlopos logikai képernyő
2400 - 25FF	CP/M paraméter blokk
2600 - 2BFF	BIOS 8502 kód
2C00 - 2FFF	VICII képernyő memória
3000 - 3C80	CCP puffer
4000 - 9BFF	szabad terület
9C00 - C9FF	BDOS átlapolt rész
CA00 - EDFF	BIOS átlapolt rész
EE00 - F3FF	BDOS rezidens rész
F400 - FBFF	BIOS rezidens rész
FC00 - FCFF	megszakítási vektorok
FD00 - FDFF	rendszer paraméterek a közös részben
FE00 - FEFF	lemez puffer
FF00 - FFFF	MMU és átkapcsoló rutinok

Az 1. szeleten levő memóriakiosztás felel meg a CP/M szabványnak. Ennek megfelelően a rendszer a 0. lapot a következőképpen használja:

Az 1. szelet használata

Cím	Leírás
0000H-0002H	JP WBOOT utasítás (JP F403
0003H	foglalt
0004H	foglalt
0005H-0007H	JP BDOS utasítás (JP EE06 ha nincs aktiv RSX)
0008H-003AH	RST utasításoknak fenntartva
003BH-004FH	szabad terület
0050H	annak a lemezegységnek az azonosítója, ahonnan a programot betöltöttük
0051H-0052H	mutató: az első FCB jelszavára = 0, nem adtunk meg jelszót
0053	az első FCB-hez tartozó jelszó hossza = 0, nem adtunk meg jelszót
0054-0055	mutató: a második FCB jelszavára = 0, nem adtunk meg jelszót
0056H	a második FCB-hez tartozó jelszó hossza = 0, nem adtunk meg jelszót
0057H-005BH	szabad terület
005CH-007BH	File Control Block 1
006CH-007BH	File Control Block 2 Ha két file-t is megadunk, akkor a FCB-ket át kell másolnunk, mert az első vége fedi a második elejét
007CH	az FCB1 rekordszáma
007DH-007FH	File Control Block készítése a közvetlen írású rekord sorszáma
0080H-00FFH	128 byte-os puffer terület lemezműveletekhez
0100H-	Felhasználói programterület

11.3 A CP/M munkaterületei

A CP/M+ a lemezekkel való adatcsere biztosításához az ún. FCB-ket (file control blokk) használja. Az FCB-k alakja azonos a lemezen a katalógusbejegyzésekkel.

A rendszerváltozók karbantartására a CP/M külön memóriarészt tart fennt, ez az SCB (system control blokk), amelyik 100 byte-ja az elsődleges kommunikációs terület a rendszer egyes elemei között.

11.3.1 Az SCB felépítése

Az SCB egyes elemeit a BDOS funkciók automatikusan átállítják. A 49-es funkcióhívással azonban közvetlenül is lekérdezhetjük/átállíthatjuk ezeket az értékeket. Az alábbiakban a **csak olvashatónak (RO)** nevezett változók felülírása azonban tönkreteheti a rendszert.

00-04	RO	A rendszer számára fenntartott
05	RO	BDOS verziószám
06-09		A felhasználó számára fenntartott hely, szabadon lehet használni.
0A-0F	RO	A rendszer számára fenntartott
10-11		Program visszatérési kód. Ez a program használható a hiba miatti programmegállás jelzésére. A BDOS 108-as funkcióval közvetlenül írható/olvas ható.
12-19	RO	A rendszer számára fenntartott
1A		A konzol sor szélessége. Ez az érték általában 79, de a DEVICE parancs segítségével megváltoztatható.
1B	RO	A konzol pillanatnyi oszlop pozíciója
1C		A konzol lapmérete. Ez általában 24, de a DEVICE paranccsal ez megváltoztatható.
1D-21	RO	A rendszer számára fenntartott
22-2B		A CP/M+ BIOS 12 fizikai eszközt képes. A fenti byte-ok azt mutatják, hogy a CP/M+ logikai eszközei mely fizikai perifériákhoz tartoznak. A logikai-fizikai hozzárendelést egy 16 bites szó adja meg. A legmagasabb bit a 0-ás fizikai eszköznek felel meg. A 4. bit a 11-es sorszámú fizikai eszközt jelenti. A 3-0 biteknek nincs jelentése. Két byte-osával az alábbi sorrendben következnek az összerendelések: CONIN, CONOUT, AUXIN, AUXOUT, LSTOUT.
2C		Lapozási mód jelzője. Ha ez az érték nullától különböző, akkor a képernyőre írás folyamatos, ha nem, akkor az SCB 1CH byte-jában levő számú soronként megáll a kiírás. Az indítás és megállítás a <CTRL-S>, illetve a <CTRL-Q> billentyűkkel történik.
2D	RO	A rendszer számára fenntartott
2E		Megadja, hogyan működik a <CTRL-H> karakter. Ha 0, akkor a <CTRL-H> visszalép és töröl. Ha 0FFH az értéke, akkor a billentyű töröl, de nem lép vissza, hanem újból kiírja a törölt karaktert.
2E		Megadja, hogyan működik a <DEL> karakter. Ha 0FFH, akkor a <DEL> visszalép és töröl. Ha 0 az értéke, akkor a billentyű töröl, de nem lép vissza, hanem újból kiírja a törölt karaktert.
30-32	RO	A rendszer számára fenntartott
33-34		Konzol mód jelzése. A lehetséges üzemmódokat a 109-es BDOS funkció leírásánál adjuk meg.
35-36	RO	A rendszer számára fenntartott

37		A sztring végét jelző karakter. (Lásd a 110-es funkciót!)
38		Listázást jelzi. Ha 0, a konzol output nem jelenik meg a listázó periférián. Ha 1, akkor igen.
39-3B	RO	A rendszer számára fenntartott
3C-3D	RO	A jelenlegi DMA terület címe. Ennek értéket csak a 26-os BDOS funkció segítségével állítsuk! A CCP a DMA területet 0080H-ra állítja be.
3E	RO	Az aktuális lemezegység (A=0, B=1, stb.).
3F-43	RO	A rendszer számára fenntartott
44	RO	Az aktuális felhasználói szám. A lehetséges értékek 0-15.
45-49	RO	A rendszer számára fenntartott
4A		A BDOS többszektoros átviteleknél egyszerre ennyi szektort visz át.
4B		A BDOS hibamódjának kódja. Jelentését lásd a 45-ös funkcióhívás leírásánál.
4C-4F		Keresési sorrend. A SETDEF paranccsal beállított lemez keresési sorrendet határozza meg. Az első byte az elsőnek, a második a másodiknak stb. keresendő lemezegység kódját tartalmazza. Az aktuális lemezegység = 0, A=1, B=2, stb. Ha kevesebb, mint 5 egységen kell keresni, akkor a többi byte értéke 0FFH.
50		Ideiglenes file-okat tároló lemezegység azonosítója. 0=aktuális, A=1, B=2 stb.
51	RO	Hibás lemezegység. Annak a lemezegységnek az azonosítója, ahol utoljára hibát érzékelt a BDOS.
52-56	RO	A rendszer számára fenntartott
57	RO	Ha a 7. bit magas, akkor a rendszer bővített hibaüzeneteket küld.
58-59		Dátum = az 1978 január 1-e óta eltelt napok száma.
5A		Óra (binárisan kódolt decimális szám, két decimális jegy).
5B		Perc.
5C		Másodperc.
5D-5E	RO	Közös memória kezdőcíme.
5F-63		A rendszer számára fenntartott

Az FCB felépítése

A CP/M+ a lemezen levő katalógusban tünteti fel, hogy milyen file-ok vannak a lemezen ,s azok hol helyezkednek el. A katalógus 32 byte-os bejegyzéseket tartalmaz, amelynek első byte-ja dönti el, hogy mire vonatkozó bejegyzést tartalmaz.

Ha az első byte értéke 0-15, akkor az a vele megegyező felhasználói számú file-ra vonatkozó bejegyzés.

Ha az érték 16-31, akkor az egy, 15-tel kisebb felhasználói számú file kiterjesztett file kontroll blokkját tartalmazza (XFCB). Ez tartalmazza a file jelszavát és a védelem módját. Ilyen bejegyzés a katalógusba csak akkor lehet, ha a megfelelő funkciók segítségével létrehoztuk.

Ha az első byte 32, akkor a szóbanforgó bejegyzés a lemez címkéjét tartalmazza. Ennek része a lemez neve, a védelem módja, a létrehozás és a módosítás ideje.

Egészen speciális az olyan katalógusbejegyzés jelentése, amelyik 33-mal kezdődik. Ez csak a 128 byte-os logikai szektor utolsó 32 karakteres bejegyzése lehet. Ebben az esetben az INITDIR parancs kiadásával megengedtük az időfigyelést a lemezen, s 33-mal kezdődő katalógusbejegyzés az őt megelőző három másik bejegyzésre vonatkozó

idő adatokat tartalmazza. Egy y időadatsor 10 byte-ból áll, s tartalmazza a védelem módját jelző byte-ot is.

Az FCB utolsó 16 byte-ja a file-hoz tartozó lemezterületek logikai sorszámát tartalmazza.

A BDOS funkcióhívásoknál használt FCB-k az első byte-jukban eltérnek a katalógus bejegyzésektől: itt a felhasználói számát adják, ott a lemezegységet azonosítják. 0=aktuális, 1=A, B=2, stb. egészen P-ig. A 20-23. byte a lemezre nincs felírva.

Az FCB byte-jainak a jelentése a következő:

00	lemezegység kódja
01-08	file neve balra igazítva s szóközökkel feltöltve.
09-0B	a filenév kiterjesztése balra igazítva s szóközökkel feltöltve.
0C	a file-t leíró katalógusbejegyzések közül ez hányadik. Az értéke a 0-31 intervallumba eshet.
0E-0D	a rendszer használja.
0F	a 0C-ben megadott értékhez képest a rekordsorszám. Lehetséges értéke 0-255.
10-1F	a file-hoz tartozó file-területek logikai sorszáma.
20	az olvasó/író műveletekben az aktuális rekord sorszáma (a 0C-ben levő értékhez képest).
21-23	a közvetlen rekordműveletek sorszáma. Maximális értéke 3FFFFH lehet.

Az FCB file-név mezőjének 7. bitjeit ún. attributum (jelző) biteknek használja a rendszer. Ezekre a bitekre az alábbiak szerint hivatkozunk:

f1 f2 f3 f4 f5 f6 f7 f8 t1 t2 t3

11.4 BDOS funkcióhívások

A CP/M+ a BDOS funkcióhívások esetén az alábbi általános szabályokat követi. A C regiszter tartalmazza a funkció számát, míg a DE regiszterpár tartalmazza egy byte-t vagy szót, azt az információt, ami a művelet végrehajtásához szükséges. Egy sor esetben a DE-ben egy mutatót kell betölteni, ami az FCB-re, a DMA-ra vagy a SCB-re mutat. A BDOS funkciók az egybyte-os értékeket az A regiszterben, a szó értékeket a HL regiszterben adják vissza. Ezen túl az A és L, illetve a H és B regiszterek értéke megegyezik.

0-ás funkció: rendszer újraindítása

A funkció meghívása ekvivalens a CP/M 3 meleg startjának végrehajtásával, amit egy JMP 0000H utasítással is elérhetünk. A funkció **nem inicializálja a lemezes** alrendszert! A funkciót hívó program a következő programnak a 108-as funkció segítségével adhat át visszatérési kódot.

Példa:

```
        ldi c,00h
        all bdos
```

1-es funkció: konzol input

A funkció a konzolról egyetlen karaktert olvas, s azt megismétli a képernyőn. A program addig nem tér vissza a hívott programba, míg valamelyik input eszközről nem érkezik egy karakter.

Példa:

```
        ldi c,01h
        call bdos
        sta char
        ;
        ...
char    db 0
```

2-es funkció: konzol output

A funkció a konzolra kiírja az E regiszterben levő karaktert. Amennyiben a <CTRL-P> segítségével a nyomtatót bekapcsoltuk, akkor a karakter a nyomtatón is megjelenik.

Példa:

```
        lda char
        mov e,a
        ldi c,02h
        call bdos
        ;
        ...
char    db 0
```

3-as funkció: input az AUX: perifériáról

A funkció az AUX:-nak kijelölt perifériáról olvas be az A regiszterbe egyetlen karaktert.

Példa:

```
        ldi c,03h
        call bdos
        sta char
```

```
        ;
        ...
char db 0
```

4-es funkció: output az AUX: perifériára

A funkció az AUX:-nak kijelölt perifériára küldi el az E-ben levő karaktert.

```
        lda char
        mov e,a
        ldi c,04h
        call bdos
        ;
        ...
char db 0
```

5-ös funkció: irás a listázó perifériára

A funkció az LST: perifériára küldi el az E regiszterben levő értéket.

```
        lda char
        mov e,a
        ldi c,05h
        call bdos
        ;
        ...
char db 0
```

6-os funkció: direkt konzol I/O

A funkció segítségével lehetőség van a CONIN: perifériáról közvetlenül olvasni, anélkül, hogy a CP/M 3 ezeket értelmezné. Az olvasás módja az E regiszter tartalmától függ.

E: 0ffh Ebben az esetben a funkció az utoljára beolvasott karakter ASCII kódját adja vissza az A regiszterben, ha ez 0, akkor nem érkezett karakter.

E: 0feh A funkció nem olvas be karaktert a konzolról, csupán annak státusát kérdezi le. Ha az A regiszterben 0-t ad vissza, akkor nem érkezett karakter. Különben az A értéke 0ffh.

E: 0fdh A funkció a CONIN:-ről beolvas egyetlen karaktert, s annak ASCII értékét az A regiszterbe helyezi. A funkció addig nem tér vissza, míg az első karaktert be nem olvasta.

E: egyéb Az előzőektől eltérő esetben a BDOS feltételezi, hogy az E regiszter egy érvényes karakter ASCII kódját tartalmazza, s elküldi a CONOUT: logikai perifériára.

7-es funkció: AUX: input státuszának lekérdezése

A funkció az A regiszterben az AUXIN: logikai periféria állapotjelzőjét adja vissza. Értéke 0ffh, ha érkezett karakter, s 00h, ha nem.

Példa:

```
          ldi c,07h
          call bdos
          ora a
          jnz read$char
          ...
read$char ...
```

A vezérlés akkor kerül a read$char címkére, ha az AUXIN: készen áll egy karakter küldésére.

8-as funkció: AUX: output státuszának lekérdezése

A funkció az A regiszterben az AUXOUT: logikai periféria állapotjelzőjét adja vissza. Értéke 0ffh, ha a periféria képes karakter fogadására, s 00h, ha nem.

Példa:

```
           ldi c,08h
           call bdos
           ora a
           jnz write$char
           ...
write$char ...
```

A vezérlés akkor kerül a write$char címkére, ha az AUXOUT: készen áll egy karakter fogadására.

9-es funkció: sztring kiírása a konzolra

A funkció a DE által mutatott sztringet kiírja a CONOUT: logikai perifériára. A sztringnek egy $ jellel kell végződnie. A $ már nem kerül kiírásra.

Példa:

```
        ldi c,09h
        lxi d,massege
        call bdos
        ...
message "Itt a vege$"
```

10-es funkció: konzol formázott olvasása

A funkció a CP/M 3 input pufferében megszerkesztett karaktersorozatot olvassa be az általunk adott munkaterületre. Ennek a kezdőcímét a DE regiszterpárban kell beállítanunk. A puffer első eleme a puffer hosszát kell hogy tartalmazza. A funkció meghívása után a képernyőn megjelenik a kurzor, s a válasz beírása után a megszerkesztett sort a funkció átmásolja a pufferbe. A puffer második eleme a beolvasott karakterek számát tartalmazza, ezeket követik a beolvasott értékek.

Ha a DE regiszterpár értéke 0, akkor a rendszer a DMA pufferben egy inicializált karaktersorozatot vár. Ehhez a puffer harmadik helyétől kezdve a kezdő karaktersorozatot kell elhelyezni, egy 0 byte-tal lezárva.

Példa:

```
lxi d,buffer
ldi c,1ah
;
ldi h,massege
ldi d,buffer+2
ldi b,05h
lddr;
;
lxi d,buffer
ldi c,0ah
call bdos
...
```

```
message:        db "none",00h
buffer:
buff$length:    db 126
buff$number:    db 0
                ds 126
```

11-es funkció: konzol állapotának lekérdezése

A funkció a CP/M+ konzol perifériájának, a CONIN:-nek az állapotával tér vissza az A regiszterben. Az A értéke 01H lesz, ha van karakter a karakterpufferben. Ha nincs, akkor az A értéke a visszatéréskor 0.

Ha a konzol CTRL-C üzemmódban van, akkor a 11-es funkció csak akkor tér vissza 01H-val, ha az érzékelt karakter CTRL-C volt.

Példa:

```
ldi c,0bh
call bdos
```

12-es funkció: verzió szám lekérdezése

A funkció HL-ben a CP/M+ verziószámával tér vissza. H 0-t tartalmaz, jelezve a CP/M-et, L pedig (hexadecimális) 31-et tartalmaz, ami a BDOS file rendszer verziószáma.

13-as funkció: a lemezrendszer inicializálása

A 13-as funkció a lemezrendszert inicializálja. Ennek következtében valamennyi lemezegység írható/olvasható lesz, a default (aktuális) lemezegység az A lesz, s a DMA cím értéke 0080H.

14-es funkció: lemezegység kiválasztása

A 14-es funkció segítségével megváltoztathatjuk az aktuális lemezegységet. A funkció meghívása előtt a lemezegység azonosítóját az E regiszterbe kell elhelyezni. A=1, B=2 stb. Ezen túl a funkció a kiválasztott lemezegységben levő lemezt be is lépteti a rendszerbe.

Visszatéréskor az A regiszter értéke 0, ha nem történt hiba. Ha fizikai hiba történt, akkor a funkció befejezése a kiválasztott hibamódtól függ. Alapállapotban a konzolon megjelenik a hiba okára utaló üzenet, s a hívó program futása befejeződik. Ellenkező esetben a funkció visszatér a hívó programhoz az A regiszterben a 0FFH értékkel. H tartalmazza a hiba okát:

01 : lemez I/O hiba
04 : érvénytelen lemezegység azonosító (>16 vagy =0)

15-ös fukció: file megnyitása

A funkció meghívásakor a DE egy nem megnyitott FCB-re mutat. Az FCB 0. byte-ja a lemezegység azonosítóját, 1-11. byte-jai a nevét, 12. byte-ja a file-darab sorszámát tartalmazza.

Ha a file olvasással szemben jelszóval védett, akkor a jelszót a DMA első 8 byte-jára kell helyezni.

Ha a hívó program felhasználói száma nem 0, s ha a 0-ás felhasználói szám alatt nem találja a file-t, akkor megpróbálja a 0-ás felhasználói szám alatt megnyitni. Erre azonban csak akkor kerül sor, ha a file SYS attributumu.

Ha sikerrel megnyitottuk a file-t, akkkor a lemez katalógusában levő információk átmásolódnak a memóriában levő FCB-be. Ezen túlmenően, ha a file írásvédelemre védett volt, s nem adtuk meg a helyes jelszót, az FCB-ben az 'f7' attributum bitet a funkció magasra állítja. Hasonlóan, ha nem 0-ás felhasználói szám alatt nyitottuk meg a file-t, de az 0-ás felhasználói, SYS típusú file-nak bizonyult, akkor a rendszer az 'f8' attributum bitet is magasra állítja. Ennek következtében ezekbe a file-okba nem tudunk írni.

Visszatéréskor az A regiszter tartalma 00H, ha nem történt hiba, s 0FFH, ha nem sikerült a műveletet végrehajtani. Ilyenkor H tartalmazza a hiba igazi okát:

01 : lemez I/O hiba
04 : érvénytelen lemezegység megadás
07 : hibás jelszó
09 : ? karakter szerepelt a filenévbn

16-os funkció: file lezárása

A 16-os funkció egy megnyitott FCB-vel rendelkező file-t lezár. A rutin meghívásakor DE-nek a szóbanforgó file FCB-jére kell mutatnia. Az FCB 'f5' attributumbitje mutatja, hogy hogyan kell a file-t lezárni:

'f5'=1 : részleges lezárás
'f5'=0 : teljes lezárás

Teljes lezárás esetén a rendszer valamennyi szükséges információt visszírja a lemez katalógusába. (Ha csak írás vagy aktualizálás történt, akkor ez elmarad, hiszen a katalógus bejegyzést nem kell megváltoztani.) Teljes lezárás után a file-ba már nem tudunk sem írni, sem olvasni.

A parciális lezárás aktualizálja a katalógust (ha kell), de a file további műveletekre nyitva marad.

Visszatéréskor A=00H, ha nem történt, s A=0FFH, ha történt valamilyen hiba. Ez utóbbi esetben H tartalmazza a hiba okát:

01 : lemez I/O hiba
02 : csak olvasható lemez
04 : érvénytelen lemezegység azonosító

17-es funkció: katalógusbejegyzés keresése (első)

A funkció - összhangban nevével - a katalógusban megkeresi egy adott nevű file első előfordulását. A file nevében ? karakterek is szerepelhetnek.

Meghíváskor DE egy FCB-re kell hogy mutasson, amelyik 0. byte-ja a lemezegységet adja meg, 1-11. byte-ok pedig a keresendő file nevét. Ezen túl megadhatjuk a 12. byte értékét is, akkor a megfelelő sorszámú file-darab után keres a rendszer. Rendszerint a 12. byte értéke 0.

Ha a 0. byte helyére írunk be egy kérdőjelet, akkor a funkció a default lemezegység első bejegyzésével tér vissza.

A funkció visszatéréskor az A regiszterben helyezi el a megtalált bejegyzés DMA-n belüli sorszámát (0-3). Ha nem találta meg, akkor A értéke 0FFH és H tartalmazza a hiba okát.

Sikeres keresés esetén a DMA puffer A*32-ik pozíciójától találhatjuk meg a megfelelő katalógus bejegyzést.

Ha egy lemezre bevezettük a dátum és idő jelzését, akkor ezeket az információkat minden egyes 128-byte-os rekord 4. 32 byte-os része tartalmazza, s csak az első 3 darab 32 byte-os rész felel meg 'igazi' katalógus bejegyzésnek. Ebben az esetben a DMA puffer 97+(A*10) címétől kezdődően 9 byte tartalmazza a következő információkat:

0-3 : a létrehozás dátuma és ideje
4-7 : az aktualizálás dátuma és ideje
8 : a védelem módja

18-as funkció: katalógusbejegyzés keresése (következő)
A funkció hívása előtt végre kell hajtani a 17-es funkciót. A jelen hívás segítségével a rendszer megkeresi a következő katalógusbeli bejegyzést. A funkció a 17-es funkciónak megfelelő paraméterekkel tér vissza.

19-es funkció: file törlése
A funkció meghívásakor a DE egy FCB-re kell hogy mutasson. Az FCB-ben megadott névspecifikációval egyező file-okat (pontosabban valamennyi hozzájuk tartozó bejegyzést) törli a rendszer. A filenév tartalmazhat ? jelet is, a lemezegység azonosítójának azonban egyértelműnek kell lennie.

Az 'f5' attributumbit magasra állításával elérhetjük, hogy csak az XFCB bejegyzéseket (amelyek a dátum, idő és jelszó adatokat tartalmazzák) törölje a rendszer.
Ha bármelyik file jelszóval védett, a jelszót a DMA első 8 byte-jára kell helyezni - még a funkció meghívása előtt.

Visszatéréskor A=00H sikeres, A=0FFH sikertelen befejezés esetén. Ez utóbbi esetben H-ba helyezi a hiba okát:

01 : lemez I/O hiba
02 : csak olvasható lemezegység
03 : csak olvasható file
04 : érvénytelen lemezazonosító
07 : hibás jelszó

20-as funkció: szekvenciális olvasás
A szekvenciális olvasási funkció segítségével a file következő 128byte-os rekordjait olvashatjuk be a DMA területre. A beolvasott rekordok száma 1-128 lehet s a 44-es funkció segítségével lehet beállítani. A 20-as funkció meghívásakor DE egy megnyitott file FCB-jére mutat. A funkció - miközben beolvassa az egyes rekordokat - automatikusan növeli az FCB-ben a rekordszámot. Ha az túlcsordul, akkor megnöveli a 'file-terület' sorszámát és a rekordszámot 0-ról újból növelni kezdi.

Ha a file-t egy 15-ös megnyitási funkció után elölröl akarjuk olvasni, akkor az FCB-ben a rekordszámot 0-ra kell beállítani.

Viszatéréskor a DMA terület a beolvasott 128 byte-os rekordokat tartalmazza, az A regiszter pedig a hibakódot. Ha A=00H, akkor nem történt hiba, különben A tartalmazza a hiba valódi okát:

01 : nem létező adatot akartunk olvasni

09 : érvénytelen FCB
10 : a lemezt kicserélték
255 : fizikai hiba, lásd a H regisztert!

H lehetséges értékei:

01 : lemez I/O
04 : hibás lemezmegadás

21-es funkció: szekvenciális írás

A funkció segítségével 1-128 byte-os rekordot tudunk a DMA területről az adott FCB-ú file-ba kiírni. A funkció meghívásakor a DE-nek egy megnyitott FCB-re kell mutatnia. A kiírandó rekordok számát a 44-es funkcióval lehet beállítani.

A funkció automatikusan növeli az FCB-ben levő rekordszámot, s ha az túlcsordul, akkor automatikusan gondoskodik a file-terület számának növeléséről, s a rekordsorszám 0-ra állításáról.

Ha a file-t az elejétől akarjuk írni, akkor a rekordsorszámot ki kell nulláznunk.
Visszatéréskor az A regiszter tartalmazza a hiba okát:

00 : nem történt hiba
01 : nincs szabad katalógusbejegyzés
02 : nincs szabad adatterület
09 : érvénytelen FCB
10 : a lemezt kicserélték
255 : fizikai hiba

Ha az A regiszter fizikai hibát jelez (=255), akkor a H regiszter további információt ad a hiba okáról:

01 : lemez I/O hiba
02 : csak olvasható lemez
03 : csak olvasható file
(vagy a FCB 'f7','f8' attributum bitjei közül valamelyik magas)
04 : érvénytelen lemezegység

22-es funkció: file létrehozása

A funkció segítségével a lemez katalógusában egy új bejegyzést hozhatunk létre. A funkció meghívása előtt DE-be egy FCB mutatóját kell betöltenünk. Az FCB 0-12. byte-jait ki kell töltenünk. A filenév 'f6' attributum bitje mondja meg, hogy a file-hoz jelszót rendelünk-e, vagy sem. Ha ez a bit magas, akkor igen. Ebben az esetben a DMA első 8 byte-jára kell elhelyeznünk a jelszót s a 9. byte-ot a védelem módjának megfelelően kell beállítani. (Ezt a 102-es funkcióval végezhetjük el).

A file létrehozása nem történik meg, ha hasonló nevű file (ugyanazon felhasználói szám alatt) már létezik a lemezen. A funkció után a file megnyitott állapotban lesz, tehát írhatjuk, majd olvashatjuk is.

Ha az 'f6' attributum bitet magasra állítottuk, s a lemez címkéje a lemezre előírja a jelszó védelmet, akkor a funkció még egy XFCB-t is létrehoz, ahová a DMA első 9 byte-jának megfelelően beírja a (kódolt) jelszót és a védelem módját.

Ha a lemezre a dátum és idő jelzése üzemmód is be van kapcsolva, akkor ezek az adatok is a lemez katalógusába kerülnek.

Visszatéréskor A=00H a sikeres, A=0FFH a sikertelen befejezést jelzi. Ez utóbbi esetben H tartalmazza a hiba okát:

01 : lemez I/O hiba
02 : csak olvasható lemez
04 : nem megfelelő lemezazonosító
09 : ? jel szerepelt a file névben

23-as funkció: file átnevezése

A funkció meghívása előtt DE-nek egy FCB-re kell mutatnia. A file új nevét az FCB második 16 byte-jának kell tartalmaznia. Ha az átnevezni kívánt file jelszóval védett, akkor a jelszót a DMA első 8 byte-jában kell elhelyezni. A funkció a lemez katalógusában levő valamennyi - az adott file-ra vonatkozó - hivatkozást átnevezi.

Visszatéréskor A=00H a sikeres, A=0FFH a sikertelen befejezést jelenti. Ez utóbbi esetben H tartalmazza a hiba okát:

01 : lemez I/O hiba
02 : csak olvasható lemez
03 : csak olvasható file
04 : hibás lemezegység azonosító
07 : hibás jelszó
08 : a file már létezik
09 : ? a névben

24-es funkció: érvényes lemezegységek lekérdezése

A funkció visszatéréskor a HL regiszterpárban megadja a rendszerben levő lemezegységeket. Az L regiszter legalacsonyabb helyiértékű bitje az A, a H regiszter legmagasabb helyiértékű bitje a P lemezegységnek felel meg. A megfelelő bit magas volta jelzi, hogy az egység a rendszerben van.

25-ös funkció: aktuális lemezegység lekérdezése

A funkció A-ban az aktuális lemezegység azonosítójával tér vissza. Ennek megfelelően A-ban egy 0-15 közti érték van, ami az A-P egységeknek felel meg.

26-os funkció: a DMA terület kezdőcímének beállítása

A funkció meghívásakor DE az új DMA terület kezdőcímét kell hogy tartalmazza. A funkció végrehajtását követően a lemez adatátviteli műveletek az így beállított DMA területet használják.

A DMA terület legalább 128 byte. Hossza attól függ, hogy a 44-es funkcióval hány rekord egyidejű átvitelét jelöltük ki. Ha ez N volt, akkor a DMA terület hossza 128*N byte.

27-es funkció: lemezegység leíró vektora címének lekérdezése

A CP/M+ minden egyes lemezegységre egy leírót tart nyilván, amelyik a legfontosabb lemezműveletek adatait tartalmazza. A 27-es funkció a HL regiszterben az aktuális lemezegység leírójára mutató értékkel tér vissza.

Megjegyezzük, hogy ez a leíró a 0-ás szeleten van, ezért közvetlenül a felhasználói programok számára nem elérhető.

28-as funkció: lemez írásvédetté tétele
A funkció az aktuális lemezegységet READ-ONLY állapotba hozza. Ez azt jelenti, hogy a lemezegységre – a lemezes rendszer inicializálásáig – nem lehet írni.

29-es funkció: a csak-olvasható lemezek lekérdezése
A funkció visszatéréskor a HL regiszterpárban megadja a rendszerben levő csak-olvasható lemezegységeket. Az L regiszter legalacsonyabb helyiértékű bitje az A, a H regiszter legmagasabb helyiértékű bitje a P lemezegységnek felel meg. A megfelelő bit magas volta jelzi, hogy az egység a rendszerben van és csak olvasható.

30-as funkció: file attributumok beállítása
A funkció segítségével beállíthatjuk – egy már létező – file attributumait. A funkció segítségével az 'f1'-'f4', a 't1' (csak-olvasható), a 't2' (SYS file), a 't3' (archiv) attributumokat állíthatjuk be.

Lehetőség van az 'f6' attributumbit beállítására is, aminek azonban speciális jelentése van. Ennek az attributumnak a magasra állításával jelzi a rendszer, hogy a filenév 13. byte az utolsó szektorból a file által használt byte-ok számát tartalmazza. Ezt az értéket a rendszer nem számítja ki automatikusan, hanem valamennyi alkalmazói programnak magának kell beállítania. A file méret kiszámítására azonban a rendszer – ha az 'f6' attributum bitet beállítottuk – használja ezt az értéket. Ha a file névben az 'f6'-ot magasra állítottuk, akkor a 13. byte-ba az utolsó szektor még file-hoz tartozó byte-jaink pontos számát kell beírni.

A funkció meghívásakor DE egy FCB-re mutat, ami a szóban forgó file nevét tartalmazza, s az attributum bitek a kívánalmaknak megfelelően be vannak állítva. A funkció a katalógus összes – a file névvel megegyező – bejegyzésében átállítja az attributum biteket.

Visszatéréskor A=00H a sikeres, A=0FFH a sikertelen befejezést jelenti. Ez utóbbi esetben a H regiszter tartalmazza a hiba okát:

01 : lemez I/O hiba
02 : csak-olvasható lemez
04 : kiválasztási hiba
07 : hibás jelszó
09 : ? a filenévben

31-es funkció: lemez paraméterblokk címének lekérdezése

A 31-es funkció a HL regiszterpárban az aktuális lemezegység BIOS-rezidens DPB blokkjának, azaz a lemez paraméterblokkjának a kezdőcímével tér vissza. Ez a 0. szeleten található, ezért közvetlenül nem lehet megvizsgálni.

32-es funkció: a felhasználói szám beállítása/lekérdezése

Az E regiszterbe kell elhelyezni az új felhasználói kódot, vagy 0FFH-t, ha lekérdezni szeretnénk ezt az értéket. Viszatéréskor az A regiszterben lesz a felhasználói kód (feltéve, hogy E-ben 0FFH volt).

33-as funkció: közvetlen olvasás

A 33-as funkció segítségével a file 128byte-os logikai rekordjait olvashatjuk, de szemben a szekvenciális olvasással, nem a file-terület és az azon belüli sorszámot kell megadnunk, hanem az FCB r0,r1,r2 byte-jaiban közvetlenül az olvasni kívánt rekord 24 bites értéket. Ez 0-262143 sorszámú rekordok olvasásának felel meg. A három byte közül r0 a legalacsonyabb, r2 pedig a legmagasabb helyiértékű. A maximális rekordszám 03FFFFH-nak felel meg, ekkor r2 értéke 3.

A file közvetlen olvasásához a file-t 0-ás file-terület számmal meg kell nyitni. A 33-ás funkció meghívása előtt DE a használni kívánt FCB-re mutat, aminek r0,r1 és r2 byte-jait már beállítottuk. A funkció meghívása után a rekordok a DMA területre olvasódnak.

A beolvasott rekordok számát a 44-es funkcióval állíthatjuk be. Ha itt 1-nél nagyobb értéket adtunk meg, akkor annak megfelelő számú rekordot olvas be a rendszer.

Szemben a szekvenciális olvasással a közvetlen rekordszám nem növekszik. Ismételt hívása a 33-as funkciónak ugyanazokat a rekordokat olvassa be a memóriába.

A funkció sikeres végrehajtását az A=00H érték jelzi. Ellenkező esetben A a hiba kódját tartalmazza:

01 : nem létező adatok olvasása
03 : az adott file-terület nem zárható le
04 : keresés nem létező file-területre
06 : a közvetlen rekordszám túlcsordult
10 : a lemezt kicserélték
255 : fizikai hiba

Fizikai hiba esetén H tartalmazza a hiba okát:

01 : lemez I/O hiba
04 : érvénytelen lemezazonosító

34-es funkció: közvetlen írás

A közvetlen írás hasonló a közvetlen olvasáshoz. A funkció a közvetlen rekordszám által megadott rekordot írja a lemezre az adatokat a DMA-ból véve. Amennyiben a rekord egy olyan file-területre esik, amelyik még nincs allokálva a file-hoz, akkor ezt elvégzi.

A funkció hívása előtt a használni kívánt FCB-t a 0-ás file-területtel meg kell nyitni. DE a FCB-re mutat. Az FCB r0,r1 és r2 byte-jait megfelelően be kell állítani. Az átvitt rekordok számát a 44-es funkcióval állíthatjuk be.

Visszatéréskor A tartalmazza a hiba kódját:

00 : nincs hiba
02 : nics elérhető adatterület
03 : az aktuális file-terület nem zárható le
05 : nincs elérhető katalógus bejegyzés
06 : a közvetlen elérésű rekordszám túlcsordult
10 : a lemezt kicseréltük
255 : fizikai hiba

Az A=255 esetben H tartalmazza a fizikai hiba leírását:

01 : lemez I/O hiba
02 : csak-olvasható lemez
03 : csak-olvasható file
04 : érvénytelen lemezazonosító

35-ös funkció: file méretének kiszámítása
A funció meghívása előtt DE-nek egy FCB-re kell mutatnia. Visszatéréskor az FCB r0,r1 és r2 byte-jai a file által még nem használt 128 byte-os rekord sorszámát tartalmazzák. Ha pl. r2=4, akkor a file a lehetséges 262144 rekordot tartalmazza.

Ez az érték félrevezető, mert ha pl. a közvetlen írással a 262143. rekordot irtuk csak fel, akkor is 262144-et kapunk, holott a file egyetlen adatot tartalmaz csak.

Visszatéréskor A=00H, ha a file-t megtalálta a rendszer, s A=0FFH, ha nem. Mindkét esetben H értéke 0. Ha fizikai hiba történt, akkor H 0-tól különböző értéket tartalmaz:

01 : lemez I/O hiba
04 : érvénytelen lemezegység azonosító

36-os funkció: közvetlen rekordszám beállítása
A funkció az FCB file-terület és rekordsorszám byte-jainak értékéből kiszámítja a következő rekord sorszámát, s ezt az FCB közvetlen rekordszámába írja.

Erre főleg olyankor lehet szükség, ha egy adott pontig szekvenciálisan irtuk vagy olvastuk a file-t, s azután áttérünk a közvetlen írásra, olvasásra.

A funkció DE-ben az FCB mutatóját várja.

37-es funkció: lemezegység inicializálása
A funkció segítségével adott lemezegységeket inicializálhatunk. Ennek hatására az egység írható-olvasható lesz, s nem lesz beléptetett lemez az egységben. A funkció a DE-ben várja az inicializálni kívánt egységeket. Az E legkisebb helyiértékű bitje az A, a D legnagyobb helyiértékű bitje a P egységnek felel meg. A megfelelő bit magasra állítása jelzi, hogy az egységet inicializálni kell.

38-as funkció: lemez lefoglalása
Ez MP/M funkció, a CP/M+-ban nincs megvalósítva. Ha mégis meghívjuk, az A értéke 00H lesz, jelezve, hogy a funkció sikerrel lefutott.

39-es funkció: lemez felszabadítása
Ez MP/M funkció, a CP/M+-ban nincs megvalósítva. Ha mégis meghívjuk, az A értéke 00H lesz, jelezve, hogy a funkció sikerrel lefutott.

40-es funkció: közvetlen írás allokálással

A 40-es funkció használata teljes mértékben azonos a közvetlen írás funkcióval (34-es funkció). Egyetlen különbség, hogy a funkció az esetleg még nem létező rekordokat is létrehozza, s azokat 00H byte-tal tölti fel.

41-es funkció: rekord ellenőrzése és írása

Ez MP/M II funkció, a CP/M+-ban nincs megvalósítva. Ha mégis meghívjuk az A értéke 0FFH, a H értéke 0 lesz.

42-es funkció: rekord zárolása

Ez MP/M II funkció, a CP/M+-ban nincs megvalósítva. Ha mégis meghívjuk az A értéke 00H lesz, jelezve, hogy a funkció sikerrel lefutott.

43-as funkció: rekord felszabadítása

Ez MP/M II funkció, a CP/M+-ban nincs megvalósítva. Ha mégis meghívjuk az A értéke 00H lesz, jelezve, hogy a funkció sikerrel lefutott.

44-es funkció: többszektoros adatátvitel beállítása

A funkció E-ben az egyszerre átviendő szektorok számát várja. Ezt követően valamennyi írási/olvasási művelet egyszerre ennyi 128byte-os rekordot mozgat.

E értékének az 1-128 intervallumba kell esnie. Ha nem, akkor visszatéréskor A=0FFH lesz.

A CCP az egyszerre átviendő szektorok számát 1-re állítja be.

45-ös funkció: BDOS hibamód beállítása

A funkció meghívása előtt E-ben a kívánt hibamód kódját kell beállítani. Ezek a kódok a következők:

0FFH : visszatérési üzemmód
0FEH : kijelzési és visszatérési üzemmód
egyéb : kijelzési üzemmód (default)

A BDOS indításkor kijelzési üzemmódban dolgozik. Ez azt jelenti, hogy hiba esetén a CONOUT: periférián a hiba okára utaló üzenet jelenik meg, s a hívó program futása megszakad.

Visszatérési üzemmódban a BDOS nem küld hibaüzenetet, s visszatér a hívó programba (A és H értékét megfelelően beállítva).

Kijelzési és visszatérési üzemmódban a képernyőn kapunk üzenetet, de a BDOS visszatér a hívó programhoz.

46-os funkció: szabad lemezkapacitás lekérdezése

A funkció a lemez szabad kapacitását adja meg 128byte-os rekordokban mérve. A funkció meghívásakor E tartalmazza a lemezegység kódját (0=A,1=B,...,15=P).

Visszatéréskor A DMA első 3 byte-ja egy hárombyte-os bináris számot ad, ez a szabad rekordok száma. Az első byte a legalacsonyabb, a harmadik a legmagasabb helyiértékű byte.

Sikeres végrehajtás esetén A=00H, egyébként A=0FFH. Ez utóbbi esetben H tartalmazza a hiba kódját:

01 : lemez I/O hiba
04 : érvénytelen lemezazonosító

47-es funkció: program töltése
A funkció a DMA terület elején - 00H byte-tal lezárva - egy CP/M+ parancsot vár, amit a CCP végre is hajt. Ha az E regiszter értéke 0FFH, akkor a CCP az aktuális lemezegységet és a felhasználói számot a 47-es funkciót hívó programból veszi át. Ha nem, akkor a saját default értekeit használja.

A betöltött programnak a 108-as funkcióval egy kétbyte-os értéket adhatunk át.
A funkció értelemszerűen befejezi a program futását, ezért a keletkező hibákat a CCP kezeli le.

48-as funkció: pufferek kiürítése
A funkció hatására a rendszer az összes puffer tartalmát kiírja a lemezre. Ha pl. ellenőrizni akarjuk, hogy a lemezre azt írtuk-e, amit akartunk, akkor ezt a parancsot mindig ki kell adnunk.

A sikeres végrehajtást az A=00H érték jelzi. Ellenkező esetben A=0FFH és H tartalmazza a hiba okát:

01 : lemez I/O
02 : csak-olvasható lemez
04 : érvénytelen lemezegységazonosító

49-es funkció: SCB lekérdezése/beállítása
A funkció segítségével a SCB byte vagy szó értékeit módosíthatjuk, illetve lekérdezhetjük. A funkció meghívásakor a DE egy SCB paraméter blokkra mutat, melynek felépítése a következő:

```
SCPPB:
        DB ELTOLAS      ; az SCB-n belüli kezdő pozíció
        DB MOD          ; 0FFH = byte beállítása
                        ; 0FEH = szó beállítása
                        ; 00H  = byte lekérdezése
        DW ERTEK        ; a beállítandó érték (ha beállítás)
```

A funkció működése a paraméter blokk második byte-jától függ. Ha beállításról van szó, akkor a SCB megfelelő byte-jába, vagy szavába bemásolja a paraméterblokkban megadott értéket. Ha lekérdezésről van szó, akkor a megfelelő byte az A regiszterbe kerül, míg az azon a helyen levő szó a HL regiszterpárba.

50-es funkció: BIOS hívás
A funkció hívásakor DE egy BIOS paraméterblokkra kell, hogy mutasson. A BIOS paraméterblokk felépítése a következő:

```
BIOSPB:
        DB FUNC         ; BIOS funkció száma
        DB AREG         ; az A regiszter tartalma
        DW BCREG        ; a BC regiszterpár tartalma
        DW DEREG        ; a DE regiszterpár tartalma
        DW HLREG        ; a HL regiszterpár tartalma
```

A funkció az A, BC, DE illetve HL regiszterek értékét beállítja, s meghívja a megfelelő BIOS funkciót.

59-es funkció: átlapolt program töltése

A funkció a DE-ben megadott FCB-nek megfelelő file-t betölti a memóriába. A program abszolút vagy relokálható lehet. Ez utóbbit a PRL típusáról ismeri fel a rendszer. A hívó programnak legalább egy RSX rezidens bővítést kell tartalmaznia, különben a LOADER modul nincs a memóriában.

Az FCB-t a funkció meghívása előtt meg kell nyitni. A töltési cím a közvetlen rekordszám első két, r0 és r1 byte-ja. A LOADER modul hibajelzést ad, ha a töltési cím kisebb, mint 0100H, vagy felülírná önmagát.

Visszatéréskor A és H tartalmazza a hiba okát:

A	H	
0FFH	00H :	nincs meg a LOADER modul
0FEH	00H :	töltési cím <0100H, vagy a LOADER felülírná önmagát
0FFH	xxH :	ugyanaz, mint a 20-as, szekvenciális olvasás funkció esetén

60-as funkció: RSX bővítés hívása

A funkció meghívása előtt DE egy RSX paraméterblokkra kell hogy mutasson. Az RSX paraméterblokk felépítése a következő:

```
RSXPB:
        DB FUNC        : funkció szám
        DB DARAB       : a szó paraméterek száma
        DW PARAM1      : első paraméter
        . . .
        DW PARAMn      : utolsó paraméter
```

A funkcióhívás végigfut a teljes RSX bővítési láncon. Ha egyáltalán nincs RSX bővítés, akkor A és L értéke 0FFH lesz.

A 0-127 függvényhívások a BDOS számára fenntartottak. A felhasználói szoftverek a 128-255 hívásokat használhatják.

98-as funkció: ideiglenesen allokált blokkok felszabadítása

A funkció az összes beléptetett lemezen felszabadítja az ideiglenesen allokált blokkokat. Ezek azok a blokkok, amelyek valamelyik file-hoz tartoznak, de ez a tény még a katalógusban nincs rögzítve.

A funkció sikeres végrehajtásakor A=00H, különben A=0FFH. Ilyenkor H=04H, jelezve, hogy lemez I/O hiba történt.

99-es funkció: file csonkolása

A funkció meghívásakor DE egy FCB-re mutat. A funkció a közvetlen rekordszámnál levágja a file-t, az annál nagyobb sorszámú rekordok helyét felszabadítja.

Ha a file jelszóval védett, akkor a DMA első 8 byte-ján a jelszót is meg kell adnunk a funkció hívása előtt.

A funkció meghívásakor az adott file nem lehet nyitott.
Sikeres végrehajtás esetén A=00H, ellenkező esetben A=0FFH, s a H regiszter tartalmazza a hiba okát:

01 : lemez I/O hiba
02 : csak-olvasható lemez
03 : csak-olvasható file
04 : érvénytelen lemezegység azonosító
07 : hibás jelszó
09 : ? a filenévben

100-as funkció: lemez címkéjének megadása

A funkció segítségével adhatjuk meg a lemez címkéjét. A funkció meghívása előtt a DE egy FCB-re kell hogy mutasson. Ez tartalmazza a lemez címkéjének nevét, a lemezegység azonosítóját. Az FCB 12. byte-ja a további paramétereket adja meg. Ezek a 12. byte megfelelő bitjeinek magasra állításával aktivizálhatók:

7. : védelem bekapcsolása
6. : olvasási művelet esetén dátum és idő beállítása
5. : módosítás esetén dátum és idő beállítása
4. : létrehozás esetén dátum és idő beállítása
0. : új jelszó adása a címkéhez

Ha a lemez címkéje jelszóval védett, a jelszót a DMA első nyolc byte-jára kell elhelyezni. Ha az FCB 12. byte-jának 0. bitjét magasra állítottuk, akkor a DMA második 8 byte-jára kell az új jelszót beírni.

Visszatéréskor az A=00H érték jelzi, hogy a funkció sikeresen befejeződött. Ha nincs hely a címke felírására, vagy az idő és dátum felírást követeljük, de a lemez nem tartalmaz SFCB-ket, akkor A=0FFH lesz, miközben H=00H. Ha fizikai hiba történt, akkor A=0FFH ugyancsak, de H egy nem nulla értéket tartalmaz:

01 : lemez I/O hiba
02 : csak-olvasható lemez
04 : érvénytelen lemezegység azonosító
07 : hibás jelszó

101-es funkció: lemez cimbyte-jának lekérdezése

A funkció segítségével a 100-as funkciónál az FCB 12. byte-jaként megadott cimbyte-ot lehet lekérdezni. A funkció meghívásakor E-be a lemez azonosítóját (A=0,B=1,...,P=15) kell megadni. Visszatéréskor A-ba kerül a szóbanforgó byte, amelynek egyes bitjei (ha magasak) az alábbiakat jelentik:

7. : jelszóvédelem bekapcsolva
6. : olvasás esetén dátum és idő jelzés
5. : módosítás esetén dátum és idő jelzés
4. : létrehozás esetén dátum és idő jelzés
0. : a lemezen van címke

Sikeres végrehajtás esetén A=00H, különben A=0FFH és a H regiszter tartalmazza a hiba kódját:

01 : lemez I/O hiba
04 : érvénytelen lemezegység azonosító

102-es funkció: a file dátum és idő adatainak olvasása

A funkció meghívásakor DE egy FCB-re mutat. A funkció az FCB 12. és 24-32. byte-jait tölti fel információval. A 12. byte magasra állított bitjeinek a jelentése a következő:

7. : olvasás ellen védelem
6. : írás ellen védelem
4. : törlés ellen védelem

Ha a 12. byte =00H, akkor a file egyáltalán nem védett.

A 24-27., illetve a 28-31. byte-ok a kreálás vagy elérés, illetve a módosítás utolsó idejét (= dátum és idő) mutatják. ennek alakját a 104-es funkciónál írjuk le.

Sikeres végrehajtás esetén A=00H. Ha a file-t nem találta a rendszer, akkor A=FFH, de H=00H. Ha fizikai hiba történt, akkor H 00H-tól eltérő értéket tartalmaz:

01 : lemez I/O
04 : érvénytelen lemezazonosító
09 : ? a filenévben

103-as funkció: file XFCB írása

A funkció segítségével XFCB-t írhatunk a lemezre, ami a file védelmi módját, jelszavát tartalmazza. A DE-nek a szóban forgó file FCB-jére kell mutatnia. A 12. byte magasra állított bitjei határozzák meg a védelem módját:

7. : olvasási védelem
6. : írásvédelem
5. : törlés elleni védelem
0. : új jelszó megadása

Ha a file jelszó védett volt, a jelszót a DMA első nyolc byteján kell elhelyezni. Ha a 0. bitet magasra állítottuk, akkor az új jelszót a DMA terület második 8 byte-ján kell elhelyezni.

Ha a funkció sikerrel befejeződött, akkor az A regiszter értéke visszatéréskor 0H. Ha a lemezen nincs címke, az FCB-t nem találta meg a funkció, nem volt elég hely az XFCB beírásához, vagy a címkében nem volt bekapcsolva a védelem, akkor A=0FFH és H=00H. Ha fizikai hiba történt, akkor H értéke a következő:

01 : lemez I/O hiba
02 : csak-olvasható lemez
04 : érvénytelen lemezegység azonosító
07 : hibás jelszó
09 : ? a filenévben

104-es funkció: dátum és idő beállítása

A funkció beállítja a CP/M+ belső óráját. DE az adatokat leíró területre mutat. Ez összesen 4 byte-ot tartalmaz, amelyek jelentése az alábbi:

0-1. byte : dátum
2. byte : óra
3. byte : perc

A dátum 16 bites egész számnak tekintendő, s az 1978 január 1-je óta eltelt napokat jelenti.

A funkció a másodperceket nullázza.

105-ös funkció: dátum és idő lekérdezése

A funkció lekérdezi a CP/M+ belső óráját. DE az adatokat leíró területre mutat. Ez összesen 4 byte-ot tartalmaz, amelyek értékét a funkció beállítja. Ez egyes byte-ok jelentése az alábbi:

0-1. byte : dátum
2. byte : óra
3. byte : perc

A dátum 16 bites egész számnak tekintendő, s az 1978 január 1-je óta eltelt napokat jelenti.

A funkció a másodperceket az A regiszterben adja vissza.

106-os funkció: általános jelszó megadása

A funkcióval lehetőség nyílik általános jelszó megadására. Jelszóval védett file-ok esetén a rendszer a DMA első 8 byte-ját és az általános jelszót vizsgálja meg. Ha ezek bármelyike azonos a file XFCB-ban levő jelszavával, a file-t használhatjuk.
A funkció meghívásakor a DE a jelszó első byte-jára kell hogy mutasson.

107-es funkció: sorozatszám lekérdezése

A funkció meghívásakor a DE-nek egy 6 byte-os munkaterületre kell mutatnia, ahová a funkció beírja a CPM+ sorozatszámát.

108-as funkció: program visszatérési kód lekérdezése/beállítása

A CP/M+ megengedi, hogy minden egyes program - mielőtt megáll - egy kétbyte-os visszatérési kódot adjon át a rendszernek, amit azután a többi program lekérdezhet.

A funkció működése a DE tartalmától függ. Ha DE=0FFFFH, akkor az lekérdezésnek számít, s a funkció HL-ben az utoljára kapott visszatérési kódot adja vissza. Ha DE értéke ettől eltér, akkor DE értéke lesz a visszatérési kód.

Az egyes értékek jelentése az alábbi:

0000-FEFF	sikeres befejezés
FF00-FFFE	sikertelen befejezés
0000	a CCP ezt az értéket állítja be
FF80-FFFC	fenntartott
FFFD	a program BDOS hiba miatt állt meg
FFFE	a program CTRL-C megnyomása miatt állt meg

109-es funkció: konzol mód lekérdezése./beállítása

A funkció segítségével lekérdezhetjük, illetve beállíthatjuk a konzol módot. Ha DE=0FFFFH a funkció meghívásakor, akkor az lekérdezésnek számít, s HL-ben kapjuk vissza a konzol mód kódját.
Ha DE ettől eltérő értéket tartalmaz, az beállításnak számít. A DE egyes bitjeinek a jelentése ekkor a következő:

0. bit :	=1 CTRL-C-t jelzi csak a 11-es funkció
	=0 a 11-es funkció normálisan működik
1. bit :	=1 letiltja a CTRL-S, CTRL-Q használatát
	=0 megengedi a CTRL-S, CTRL-Q használatát
2. bit :	=1 TAB bővítést letiltja, CTRL-P-t és a nyomtatóra írást úgyszintén
	=0 normál üzemmód
3. bit :	=1 letiltja a CTRL-C-vel való megállítást
	=0 megengedi a CTRL-C-vel való megállítást
8-9. bit:	RSX-k kezelése. Ezek a bitek mondják meg, hogy válaszoljanak az RSX bővítések a konzoligényekre:
	= 0 0 feltételes mód
	= 0 1 hamis állapot
	= 1 0 igaz állapot
	= 1 1 átirányítás

A CCP a konzol módot 0-ra állítja be.

110-es funkció: elválasztójel lekérdezése/beállítása

A 9-es funkció a karakterek nyomtatását a 110-es funkcióval beállított elválasztójelig végzi. Ezt az értéket a CCP a $ jelre állítja be, de ennek a funkciónak a használatával lehetőség van ennek módosítására.

Ha DE=0FFFFH, akkor az lekérdezésnek számít, s az A-ban kapjuk vissza az érvényes elválasztó jelet.

Ha DE értéke ettől eltér, akkor az E regiszterben levő karakter lesz az aktuális elválasztó jel.

111-es funkció: blokk nyomtatása

A funkció meghívása előtt DE-nek egy karakter kontrol blokkra kell mutatnia. A karakter kontrol blokk felépítése a következő:

```
CCB:
        DW CIM   ; az első kiírandó karakter címe
        DW HOSSZ ; a kiírandó karakterek száma
```

A funkció a CCB-ben megadott karaktereket kiírja a CONOUT: perifériára.

112-es funkció: blokk listázása

A funkció meghívása előtt DE-nek egy karakter kontrol blokkra kell mutatnia. A karakter kontrol blokk felépítése a következő:

```
CCB:
        DW CIM   ; az első kiírandó karakter címe
        DW HOSSZ ; a kiírandó karakterek száma
```

A funkció a CCB-ben megadott karaktereket kiírja a LST: perifériára.

11.5 A HELP és a KEYFIG programok

A Commodore 128-hoz adott CP/M+ rendszerlemez további két programot tartalmaz, amelyik gyakran hasznos lehet.

HELP

A HELP használatához a használt lemezen két file-nak is kell lennie: az egyik a HELP.COM, a másik a HELP.HLP file. A CP/M+ használatára vonatkozó információkat valójában a HELP.HLP file tartalmazza.

Szintaxis:
HELP {téma} {résztéma1 ... résztéma8} { **[NOPAGE|LIST]** }

HELP [EXTRACT]
HELP [CREATE]

A HELP első formájában a segítő információkat olvashatjuk. A megfelelő téma képernyőn történő megjelenítése után az utolsó sorban

HELP >

üzenet jelenik meg a képernyőn. Lehetőségünk van újabb téma és résztémaválasztással más témákról információt kapni. Ha a beírás elé tizedespontot (.) teszünk, akkor azt úgy értelmezi a program, hogy a jelenlegi témák résztémáit adtuk meg, s onnan keresi az általunk beírt résztémákat.

Ha például a HELP ED parancs kiadása után a .COMMANDS-t válaszoljuk, akkor az ekvivalens a HELP ED COMMANDS kiadásával.

Ha az előbbire csak COMMANDS-szal válaszolunk (tizedespont nélkül), akkor hibaüzenetet kapunk, mert ilyen (a legfelsőbb szinten levő) téma nincs.

A HELP másik két alakja saját segítő képernyők elkészítésére szolgál.

EXTRACT Az opció segítségével a HELP.HLP file-ból előállíthatunk egy HELP.DAT nevű file-t, amelyik a segítő képernyők szövegeit tartalmazza. Ezt a file-t bármelyik CP/M+ alatt üzemelő szövegszerkesztővel módosíthatjuk.

A szöveg szerkesztésénél arra kell ügyelnünk, hogy a résztémákat mindig egy

///n<résztéma>

karaktersorozatnak kell bevezetnie. A három darab / jelről ismeri fel a HELP, hogy hol kezdődnek résztémák. n-nek 1 és 9 közé kell esnie, s a résztéma szintjét határozza meg.

A résztémák megírásánál ezen túlmenően még tekintettel kell lenni arra, hogy azokat ABC sorrendben, s mindig a közvetlen felettük levő téma után kell a szövegfile-ba behelyeznünk.

CREATE Az opció a HELP.DAT file-ból előállít egy új HELP.HLP file-t. Ezután a HELP egyszerű használata már az új segítő képernyőket adja.

KEYFIG

A KEYFIG program lehetőséget ad arra, hogy a billentyűzet használatát módosítsuk. Lehetőség nyílik a billentyűzethez más ASCII kódok hozzárendelésére, illetve bizonyos ASCII kódú billentyűk esetén azokhoz karaktersorozatot rendelhetünk hozzá. A Turbo Pascal-ról szóló részben ajánlunk egy billentyűzet kiosztást a program használatához.

A KEYFIG segítségével módosíthatjuk a logikai-fizikai színmegfeleltetést is. Erre főleg olyenkor van szükség, ha a monitorunk minősége nem megfelelő, s valamely program színkiosztása gondot okoz.

A KEYFIG segítségével a billentyűzet és a színek új definícióit visszaírhatjuk a lemezre a CPM.SYS file-ba. Ennek hatására a következő rendszertöltéskor automatikusan az új definíciók lépnek életbe.

12. fejezet

BIOS funkcióhívások

12.1 BIOS funkcióhívások felhasználása

A CP/M rendszerek előnye éppen az, hogy pontosan meghatározzák a rendszernek a hardverfüggő részét, s hogy azzal a rendszer többi része hogyan kell hogy kommunikáljon. Ez a BIOS, ami teljes egészében gépfüggő. Tartalmaznia kell valamennyi hardver függő funkciót ahhoz, hogy a CP/M többi része kommunikálni tudjon vele. Szerencsére a géppel együtt szállított CP/M+ rendszerlemez már készen tartalmazza ezeket a részeket, s nem a felhasználónak kell ezeket megírnia. A BIOS az I/O funkciók egy részét úgy hajtja végre, hogy bizonyos paraméterek beállítása után kikapcsolja a Z80-as processzort és visszakapcsolja a 8502-es processzort. Az elvégzi a szükséges feladatot, majd visszaadja a vezérlést a Z80-asnak. A programnak ezt a részét 8502BIOS-nak hívjuk, utalva arra, hogy a rendszerfunkciókat egy másik processzor végzi el.

A BIOS 33 vektorból álló ugrótáblát tartalmaz (hasonlóan mint a KERNAL). Ezek segítségével lehet az egyes funkciókat végrehajtani.

A BIOS funkciói sorrendben a következők:

Sorszám	Utasítás	BIOS táblázat	Belépési pont	Jelentés
0	JP BOOT	F400	CA00	hidegindítás
1	JP WBOOT	F403	F46C	melegindítás
2	JP CONST	F406	F56F	konzol állapotának leolvasása
3	JP CONIN	F409	F588	konzolról karakter beolvasása
4	JP CONOUT	F40C	F40A	konzolra karakter kiírása
5	JP LIST	F40F	F4E6	LST:-re karakter kiírása
6	JP AUXOUT	F412	F4E0	AUX:-ra karakter kiírása
7	JP AUXIN	F415	F58D	AUX:-ről karakter beolvasása
8	JP HOME	F418	CA6A	A lemezegység a 0. sávra áll
9	JP SELDSK	F41B	CA3C	lemezegység kiválasztása
10	JP SETTRK	F41E	CA6D	sáv számának beállítása
11	JP SETSEC	F421	CA73	szektor számának beállítása
12	JP SETDMA	F424	CA79	a DMA kezdőcímének beállítása
13	JP READ	F427	CA90	az adott blokk beolvasása
14	JP WRITE	F42A	CAA5	az adott blokk kiírása
15	JP LISTST	F42D	F50D	az LST: állapotának lekérdezése
16	JP SECTRN	F430	CA85	szektor fizikai számának kiszámítása
17	JP CONOST	F433	F503	a konzol írási állapotának lekérdezése
18	JP AUXIST	F436	F574	az AUX: olvasási állapotának lekérdezése
19	JP AUXOST	F439	F508	az AUX: írási állapotának lekérdezése
20	JP DEVTBL	F43C	F4D2	karakteres I/O eszközök leírás (kezdőcím)
21	JP DEVINI	F43F	F737	inicializálja a karakteres I/O eszközöket
22	JP DRVTBL	F442	F4D6	a lemezegységek leírása (kezdőcím)
23	JP MULTIO	F445	CAC6	logikailag összefüggő blokkok száma
24	JP FLUSH	F448	CACA	a puffer fizikai írása/olvasása
25	JP MOVE	F44B	F8A0	memóriatranszfer
26	JP TIME	F44E	F82E	az idő olvasása/írása

27	JP SELMEM	F451	F60F	a memória szelet kiválasztása
28	JP SETBNK	F454	CA81	szelet megadása DMA-hoz
29	JP XMOVE	F457	F8AB	szeletek közti MOVE előkészítése
30	JP USERF	F45A	F65B	Commodore CP/M+ felhasználói funkció
31	JP RESERV1	F45D	0000	fejlesztés alatt
32	JP RESERV2	F460	0000	fejlesztés alatt

Az ugrótábla elhelyezkedését onnan lehet megállapitani, hogy a 0000H-0002H címeken levő utasítás a tábla 2. helyére mutat. A CP/M szabvány szerint az ugrótábla csak laphatáron kezdődhet (a cím alsó byte-ja 0), ezért a WBOOT alsó byte-ja mindig három. A felső byte-ja pedig az ugrótábla kezdőcíme.

Az ugrótábla a C-128 esetén a közös rendszermemóriában helyezkedik el, ezért mind a 0., mind az 1. szeletből hívhatók. Vannak azonban olyan rutinok is a táblázatban, amelyek további részei már a 0. szeleten találhatóak. Ezért ezek meghívása előtt át kell kapcsolni a 0. szeletre. (Az összes Cxxx belépési pontú rutin ilyen!)

A fenti táblázatból két dolog rögtön kitűnik: a CP/M - szemben a C-128-as móddal - különbséget tesz karakteres és blokkos I/O eszközök között. (Lásd a 20. és a 22. rutint.) Ez utóbbiak esetében pedig közvetlenül a memóriában olvassa be a blokkot. A másik, hogy a BIOS ugrótábla egy sor olyan funkciót tartalmaz, amire a memória speciális használata miatt van szükség. Ezek a funkciók egyébként a C-128 KERNAL-ban is megtalálhatók.

Két egyszerű példát adunk a BIOS használatára. Mind a kettőt ki is próbálhatjuk, ha egy CP/M alatt futó gépi kódú monitorral rendelkezünk.

Képernyő törlése

```
4000    ld      c,1b        ; ESC betöltése c-be, majd
        call    f4da        ;   kiírása
        ld      c,3a        ; : kódjának betöltése c-be, majd
        call    f4da        ;   kiírása
        ret                 ; visszatérés a monitorhoz
```

A rutin az ESC : karaktersorozatot küldi el a konzolra, aminek hatására az törlődik. A c4000 monitor parancs ezek után tehát törölni fogja a képernyőt.

A második példa a 0.lap elérését mutatja be a SELMEM használatával. Ehhez a gépi kódú rutinnak a közös memória részben kell lennie. Legkézenfekvőbb a lemezműveletek puffer területét használni. Ez 256 byte az FE00H címtől kezdődően.

Képernyő POKE

Az alábbi rutin a 40 oszlopos képernyőhöz tartozó 80 oszlopos logikai képernyő első byte-jába írja az u betű képernyő kódját (=15H). Ennek hatására a képernyő bal felső sarkában megjelenik az u betű. Maga a program:

```
fe00    ld      a,0         ; a 0.szelet kijelölése
        call    f60f        ;  és bekapcsolása (SELMEM)
        ld      a,15        ; a-ban az u kódja
        ld      (1400),a    ;  beírása a 80 oszlopos logikai képernyőre
        ld      a,1         ; az 1.szelet kijelölése
        call    f60f        ;  és bekapcsolása (SELMEM)
        ret
```

Abban az esetben, ha olyan programokat szeretnénk készíteni, amelyek bármely CP/M V3.0+ rendszerben üzemelnek, a fenti hívások nem megfelelők. A CP/M rendszer csak azt köti ki, hogy a 0000 címen egy JMP WBOOT utasítás található. Ebből kell kiszámítani a táblázat többi rutinjának helyét. Például a képernyő törlő rutint az alábbi formában kell használni:

```
4000    ld      c,1b        ; ESC a c regiszterben
        call    4010        ;  c kiírása a konzolra
        ld      c,3a        ; : a c regiszterben
        call    4010        ;  c kiírása a konzolra
        ret                 ; alprogram vége

4010    push    hl          ; hl elmentése
        ld      hl,(0001)   ; hl-ben a WBOOT címe
        ld      l,0c        ; hl-ben a 4. BIOS címe (0c=3*4)
        ex      (sp),hl     ; hl régi értéke vissza és a 4.BIOS rutin
        ret                 ;  címe a veremben. ret-tel vezérlésátadás a 4.BIOS rutinra
```

A 4010 alatti alprogram a BIOS tábla kezdőcímét veszi a JP WBOOT 0000 alatti utasításból s az alsó (H) byte-ját átírja. Ezt a címet berakja a verembe s egy ret-tel erre a címre adja a vezérlést.

A BIOS címek ismerete nélkül is kínál a CP/M a BIOS funkciók hívására egy általános eljárást. Ehhez a BDOS 50. funkcióját kell használnunk. Ebben az esetben a hívás általános alakja a következő:

```
        ld bc,50
        ld de,parameter
        call 5
.....
parameter:
        db alfunkció
        db Areg
        dw BCreg
        dw DEreg
        dw HLreg
```

Szavakkal: BC-ben kell megadni a BDOS 50. funkciójának sorszámát, DE-be pedig a BIOS funkció paramétereinek leírására mutató vektort kell tölteni. A paraméter tábla 2 byte-ot és 3 szót tartalmaz. Ezek értéke sorban:

- a BIOS funkció száma
- az A regiszter tartalma
- a BC regiszter tartalma
- a DE regiszter tartalma
- a HL regiszter tartalma

Megjegyezzük, hogy ezzel a módszerrel valamennyi BIOS funkcióhívás elvégezhető.

12.2 BIOS funkcióhívások

A továbbiakban röviden összefoglaljuk a fenti BIOS rutinoknak a használatát:

0. BIOS rutin: BOOT
A rutin az összes hardver eszköz inicializálást elvégzi, beállítja a 0-ás lap változóit, betölti és elindítja a CCP-t. Csak a 0-ás szeletről hívható!

1. BIOS rutin: WBOOT
Beállítja a 0-ás lap változóit és újratölti a CCP-t.

2. BIOS rutin: CONST
Lekérdezi a konzol állapotát. Ha a konzolról érkezett karakter, akkor A=0FFH lesz, különben 00H.

3. BIOS rutin: CONIN
Beolvas egy karaktert a rendszerhez tartozó konzolokról. A rutin addig próbálkozik, mig nem sikerül egy karaktert beolvasnia. A karakter értéke az A regiszterbe ???aa.

4. BIOS rutin: CONOUT
A C regiszterben levő karaktert valamennyi konzolra elküldi. Addig vár, míg valamennyi, a rendszerben levő konzol jelzi a karakter feldolgozását.

5. BIOS rutin: LIST
A C regiszterben levő karaktert valamennyi LST: perifériára elküldi. Egészen addig vár, míg valamennyi a rendszerben levő listázó eszköz jelzi a karakter feldolgozását.

6. BIOS rutin: AUXOUT
A C regiszterben levő karaktert valamennyi AUX: perifériára elküldi. Egészen addig vár, míg valamennyi, a rendszerben levő AUX: eszköz jelzi a karakter feldolgozását.

7. BIOS rutin: AUXIN
Beolvas egy karaktert a rendszerhez tartozó valamelyik AUX: perifériáról. A rutin addig próbálkozik, mig nem sikerül egy karaktert beolvasnia. A karakter értéke az A regiszterbe kerül.

8. BIOS rutin: HOME
A kiválasztott lemezegység író/olvasó fejét az ún. 'home' pozícióba mozgatja. Ez a lemez 0. sávjára történő pozícionálást jelenti. Csak a 0-ás szeletről hívható.

9. BIOS rutin: SELDSK
Az utasítás hatására a C regiszter által kijelölt egység lesz az aktuális lemezegység. Ha az E regiszter alsó 4 bitje 0, akkor a rendszer úgy tekinti, hogy a szóban forgó lemezt először használjuk és beolvassa a lemez DPB-jét (disk paraméter blokk). HL a DPB kezdőcímét tartalmazza. Ha ez 00H, akkor a lemezegység nem található a rendszerben. Csak a 0-ás szeletről hívható!

10. BIOS rutin: SETTRK
A BC regiszterpár kijelöli a következő lemezművelet sávját (track-jét). Csak a 0-ás szeletről használható!

11. BIOS rutin: SETSEC
A BC regiszterpár jelöli ki a következő lemezművelet szektorszámát. Ennek az értéknek a logikai→fizikai konvertálás utáni értéknek kell lennie! Csak a 0-ás szeletről használható.

12. BIOS rutin: SETDMA
Beállítja a legközelebbi lemezművelet pufferének kezdőcímét. A rendszer a következő hivásig ezt a puffert használja. Csak a 0-ás szeletről használható.

13. BIOS rutin: READ
A kijelölt blokkot beolvassa a memória pufferbe. Ha olvasás közben hiba történik, a rendszer újra próbálkozik. Többszöri sikertelen kísérlet esetén az A regiszter tartalma 01H lesz. A rutin ellenőrzi, nem cserélték-e ki a lemezt. Ha igen, A értéke -1, azaz 0FFH lesz. Csak a 0-ás szeletről hívható!

14. BIOS rutin: WRITE
A kijelölt blokkba írja az adatpuffer tartalmát. Ha hiba történik, a rendszer újra próbálkozik. Többszöri sikertelen kísérlet után A értéke 01H lesz. Ha az utolsó hívás óta kicserélték a lemezt, A értéke -1, azaz 0FFH lesz.

15. BIOS rutin: LISTST
A rutin ellenőrzi a rendszerben levő LST: perifériák állapotát. Ha valamennyi kész a következő karakter fogadására, akkor A értéke 0FFH lesz, különben 00H.

16. BIOS rutin: SECTRN
A BC-ben tárolt logikai szektor számot átalakítja a megfelelő fizikai szektorszámmá, s azt HL-be helyezi. A rutin meghívása előtt DE-be kell tölteni a kódtábla kezdőcímét. Csak a 0-ás lapról használható!

17. BIOS rutin: CONOST
A rutin ellenőrzi, hogy a rendszerben levő konzolok képesek-e karakter fogadására. Ha valamennyi képes, akkor A értéke 0FFH lesz, különben 00H.

18. BIOS rutin: AUXIST
Ellenőrzi, hogy érkezett-e valamelyik AUXIN: eszközről karakter. Ha igen, akkor A értéke 0FFH lesz, különben 00H.

19. BIOS rutin: AUXOST
Ellenőrzi, hogy valamennyi AUXOUT: eszköz képes-e a következő karakter fogadására. Ha igen, akkor A értéke 0FFH lesz, különben 00H.

20. BIOS rutin: DEVTBL
A rutin HL-ben a karakter tábla kezdőcímével tér vissza.

21. BIOS rutin: DEVINI
A rutin inicializálja a C regiszter által kijelölt fizikai karakteres egységet.

22. BIOS rutin: DRVTBL
HL-ben a lemezegységeket leíró tábla kezdőcímével tér vissza. A leíró tábla 16 bites szavakból áll, amelyek a szóbanforgó lemezegység DPH-jára mutatnak. Ha ennek értéke 0, akkor a lemezegység nincs a rendszerben. Csak a 0-ás szeletről hívható!

23. BIOS rutin: MULTIO
C-ben megadható az egyszerre olvasandó szektorok száma. Egyszerre maximum 16K vihető át. Csak a 0-ás szeletről hívható.

24. BIOS rutin: FLUSH
A rutin hatására a puffer tartalma fizikailag is a lemezre íródik. A FLUSH utáni olvasás egyben fizikai olvasást is jelent. Csak a 0-ás szeletről hívható. A lehetséges értékek visszatéréskor a következők:

A	Jelentés
00H	nem történt hiba
01H	felismerhetetlen hiba
02H	csak olvasható lemezegység
0FFH	a lemezt kicserélték

25. BIOS rutin: MOVE
Memóriablokk mozgatása. A rutin meghívása előtt a DE-be be kell tölteni az első másolandó byte címét, HL-be pedig azt a címet, ahova másoljuk. BC tartalmazza a másolandó byte-ok számát. A rutin végén DE és HL az utolsó átmásolt byte régi és új helyére mutat.

26. BIOS rutin: TIME
Az óra beállítása. Ha C értéke 00H, akkor a rendszer az SCB-ben (System Control Block) levő értéket írja be az óraregiszterekbe. Ha C=0FFH, akkor az órában levő értéket a rendszer az SCB területre olvassa. A rendszer pakolt decimális formában tárolja az óra, perc, másodperc értékét. A CP/M+ a 6526-os CIA chip óráját használja. Vigyázat: a rutin megváltoztatja HL és DE értékét!

27. BIOS rutin: SELMEM
Memória szelet kiválasztása. A hívó rutinnak a közös memóriarészben kell lennie. Az A akkumulátor értéke adja meg, hogy az 1-es vagy a 0-ás memóriaszeletet használja-e a rendszer.

28. BIOS rutin: SETBNK
Beállítja, hogy a lemezműveletek puffere melyik memóriaszeleten legyen. A memóriaszelet értékét az A regiszterbe kell betölteni (0 vagy 1). Csak a 0-ás lapról hívható!

29. BIOS rutin: XMOVE
A MOVE funkció nem képes memóriaszeletek közti másolásra. Ha ezt akarjuk elérni, akkor a B és C regiszterekbe kell megadnunk a másolatot, illetve a másolandó byte-okat tartalmazó szelet sorszámát (0 vagy 1). Csak a 0-ás szeletről hívható!

30. BIOS rutin: USERF
8502 BIOS felhasználói rutin hívása. Az A regiszterben kell megadni a rutin, a H regiszterben a felhasználói funkció számát. Az egyes funkcióknak teljesen eltérő jelentése van. A felhasználói rutinok funkcióit a 12.3 részben részletesen ismertetjük.

31. BIOS rutin: RESERV1
Későbbi fejlesztésre fenntartva.

32. BIOS rutin: RESERV2
Későbbi fejlesztésre fenntartva.

12.3 BIOS felhasználói rutinok

Felsoroljuk a 30. BIOS rutin alprogramjait. Az alprogram sorszámát a BIOS hívása előtt az A regiszterben kell elhelyezni. Az esetleg szükséges további paramétereket az egyes alfunkciók leírásánál adjuk meg.

Az egyes funkciók nem kizárólag a processzor regiszterekben kapják a paramétereket, hanem a memóriából is. Ezek a következők:

Memória	Elnevezés
FD01	VIC$CMD
FD02	VIC$DRV
FD03	VIC$TRK
FD04	VIC$SECT
FD05	VIC$COUNT
FD06	VIC$DATA
FD07	CUR$DRV
FD08	FAST

0. funkció: A 0.szeletről byte olvasása.
További paraméterek: DE = memória cím
Működés: A rutin a DE-ben megadott 0.szeleten levő byte-ot a C regiszterbe helyezi.

1. funkció: A 0.szeleten levő byte írása
További paraméterek: DE = memória cím
C = beírandó byte
Működés: A rutin a 0.szelet DE címére írja a C-ben levő byte-ot. Ha ez a ROM-ban van, akkor (nem írja be és) visszatéréskor A értéke FFH (különben 0).

2. funkció: Billentyűzet olvasása.
További paraméterek: nincs
Működés: A rutin a billentyűzeten éppen lenyomott billentyűre vonatkozó információval tér vissza. B-ben a rendszer a billentyű kódtáblázatbeli relatív helyzetével tér vissza. Ha B=FFH, akkor nem talált lenyomott billentyűt. Az A regiszterben a rutin a lenyomott billentyű kódját (táblázatbeli értékét) adja vissza. HL az A-ban levő érték táblázatbeli helyére mutat. C a váltók használatára vonatkozó információt tartalmazza:

1 és 0 bitek	Jelentés
0 0	kis betű
0 1	nagy betű
1 0	shiftelt jel
1 1	C= billentyű

A kódtáblázat elejére az FD09H címen levő mutató mutat.

3. funkció: Z80 ROM rutinok végrehajtása
További paraméterek: L az alfunkció számát tartalmazza.
Működés: Az L értékétől függő rutint hajtja végre. Az egyes alfunkciók paraméterei eltérőek. Az alfunkciók száma mindig páros. Az első 80 rutin a 40, illetve 80 oszlopos képernyő kezelését végzi. Az alábbi felsorolásban csak a 80 oszlopos rutinokat soroljuk fel. A 40 oszlopos – hasonló feladatokat végző – alfunkciók sorszáma 2-vel nagyobb.

0. alfunkció: karakter kiírása. A D regiszterben levő karaktert írja ki, s a kurzort is átállítja.

4. alfunkció: kurzor beállítása. A D a sor, az E az oszlop sorszámát tartalmazza.

8. alfunkció: a kurzort egy sorral feljebb mozgatja. Ha a kurzor a képernyő tetején volt, hatástalan.

12. alfunkció: kurzort egy sorral lejjebb mozgatja. Ha a kurzor az utolsó sorban volt, a rutin a képernyőt egy sorral feljebb tolja.

14. alfunkció: nincs megírva.

16. alfunkció: a kurzort egy pozícióval balra mozgatja. Ha a képernyő bal szélén állt a kurzor, hatástalan.

20. alfunkció: kurzort egy pozícióval jobbra mozgatja. Ha a kurzor a képernyő jobb szélén volt, hatástalan.

24. alfunkció: <RETURN> végrehajtása.

28. alfunkció: törlés a sor végéig. Törli a sort a kurzortól a sor végéig. (A kurzor helyén levő karaktert is!)

32. alfunkció: törlés a képernyő végéig. Törli a képernyőt a kurzortól a képernyő végéig, beleértve a kurzor helyén levő karaktert is!

36. alfunkció: karakter beszúrása. A rutin a kurzor helyére egy szóközt szúr be, s a több karaktert jobbra lépteti. A sor végén álló karakter elvész.

40. alfunkció: karakter törlése. A kurzor helyén levő karaktert törli. A kurzortól jobbra levő karaktereket egy hellyel balra lépteti. A sor végén egy szóköz jelenik meg.

44. alfunkció: sor beszúrása. A kurzor sorába egy üres sort szúr be a rutin. A kurzor alatt levő sorok eggyel lejjebb csúsznak, s az utolsó sor elvész.

48. alfunkció: sor törlése. A kurzor sorát törli a rendszer. A kurzor alatt levő sorok eggyel feljebb csúsznak, alul egy üres sor keletkezik.

50. alfunkció: nincs megírva.

52. alfunkció: a kurzor színének beállítása. A B regiszterben megadott értéknek megfelelően állítja be a kurzor színét.

56. alfunkció: a 80 oszlopos képernyő attributumainak beállítása. Az alfunkció ki/bekapcsolja a 8563-as VDC chip attributumait. B-ben magasra kell állítani azokat a biteket, amelyeket változtatni akarunk. C-ben pedig ezek értékét kell megadni.

60. alfunkció: a 80 oszlopos képernyő memória olvasása. D-ben a sor, E-ben az oszlop számát kell megadni. A rutin B-ben a karakter kódját, C-ben az attributum memóriában levő értéket adja vissza. A megfelelő (62.) 40 oszlopos alfunkció csak az inverz alakról ad információt.

64. alfunkció: a 80 oszlopos képernyő memóriájába írás. A rutin meghívása előtt meg kell adni D,E-ben a sor,oszlop értékét. B-ben a karakter kódját, C-ben pedig az attributumát.

68. alfunkció: színek olvasása. A rutin a képernyő színeit olvassa. Az egyes regiszterek tartalma:

A - karakter színe;
B - háttér szín;
C - keret színe (csak 40-es képernyő esetén);
D - a karakter attributuma (csak 80-as képernyő esetén).

80. alfunkció: sávkonverzió.

82. alfunkció: CBM kód ellenőrzése. A rutin ellenőrzi, hogy a lemezegységben levő lemez 1. sávjának 0. szektorában ott van-e a CBM kód. Csak GCR lemezek esetén használható. Visszatéréskor a Z jelző 1, ha a rendszer C128-as lemezt használ, s 0, ha C-64-est. Az A regiszter 0, ha a lemez egyoldalas, FFH, ha kétoldalas.

84. funkció: hangjelzés adása.

96. funkció: kurzor helye a 40-es képernyőn. Az alfunkció a kurzor fizikai helyét számítja ki a 80-oszlopos logikai képernyőre helyezett ablak helyzetéből.

98. alfunkció: kurzor helyének beállítása. A rutin meghívja a 96. alfunkciót majd annak eredménye felhasználásával úgy görgeti a 40 oszlopos képernyőt a logikai 80 oszlopos képernyőn, hogy a kurzor a látható részben legyen.

100. alfunkció: 40 oszlopos képernyő aktualizálása. Ha a @off40 és old$offset változók értéke nem egyezik meg, akkor a rutin a 40 oszlopos képernyőt aktualizálja.

102. alfunkció: teljes képernyő görgetése. Az előző alfunkciók csak az éppen aktualizált sort mozgatták. Ez a rutin - szükség esetén - a teljes 40 oszlopos képernyőt újra írja.

104. alfunkció: üzenet kiírása. A rutin mind a két képernyőre kiírja azt az üzenetet, aminek első byte-jára a verem tetején levő érték mutat. Az üzenetet egy 0 byte-tal kell befejezni.

106. alfunkció: üzenet kiírása. Ugyanaz, mint az előbbi, de az üzenet elejére a DE regiszterpár mutat.

112. alfunkció: ASCII → képernyő kód konverzió. Az ASCII kódot a B regiszterben kell megadni, a képernyő kódot az A regiszterben kapjuk vissza. A rutin nem konvertálja a vezérlő karaktereket.

114. alfunkció: kurzor helyének beállítása. A rutin meghívásakor a 0. szelet 2409H címén levő kétbyte-os mutató a logikai képernyő egy helyére mutat. A rutin erre a helyre állítja be a fizikai kurzort (a 40 oszlopos fizikai képernyőn). Visszatéréskor:

HL = kurzor mutató
DE = a kurzor sorának kezdete
BC = karakterek száma a sor végéig

116. alfunkció: A 80-oszlopos képernyő kurzorát az előbbihez hasonlóan átállítja. A használandó memóriacím: 0. szelet 2411H.

118. alfunkció: szín beállítása. A rutin a 8563 VDC színeit a VIC színeire állítja be. B-ben a beállítandó szín típusának, C-ben a szín értékének kell lennie. Visszatéréskor HL a logikai színtáblára mutat. A színtípusok a következők:

B	típus
00	karakter színe;
10	háttér színe;
20	keret színe.

122. alfunkció: karakter feltöltés a 8563 memóriájában. A rutin a 8563 memóriájában maximum 256 byte-ot feltölt a D-ben levő byte-tal. Az attributum memóriát – ezzel párhuzamosan – az E-ben levő byte-tal tölti fel. A címet a veremre kell helyezni, BC a feltöltendő byte-ok számát tartalmazza.

124. alfunkció: blokk másolása a 8563 memóriájában. A rutin a 8563 memóriájában – maximum 256 byte-ot – átmásol. A meghíváskor a verem tetején kell lennie a másolandó rész kezdőcímének, a DE-ben a másolat kezdőcímének, míg BC-ben az átmásolandó karakterek számának.

126. alfunkció: karakter definiálása a 8563 memóriájában. DE az új karakter alakjának kezdőcíme a Z80 memóriájában. B adja meg az újonnan definiált karakterek számát. A verem a 8563-beli kezdőcímet tartalmazza.

4. funkció: 8502 BIOS hívások.
További paraméterek: L-ben az alfunkció számát kell megadni. Esetenként további paraméterek megadására is szükség lehet.
Működés: A 8502 BIOS hívások beállítanak néhány paramétert, majd átváltanak a 8502 processzorra és ott hajtják végre a megfelelő funkciókat.

-1. alfunkció: Az L=FFH értékű alfunkció inicializálja a C-128-at. Hatása megegyezik a RESET gomb megnyomásával.

0. alfunkció: A 8502 inicializálása. Beállítja a $0314-$0319 megszakító vektorokat, a PAL változót, lezárja a nyitott csatornákat, végül beállítja az összes olyan rendszerváltozót, ami a Z80 és 8502-es közti átkapcsoláshoz szükséges.

1. alfunkció: 1541-es olvasás. Az olvasandó blokk sáv és szektorszámát a VIC$TRACK, illetve VIC$SECTOR változókban kell megadni. A VIC$DRV változó alsó négy bitje az alábbi megnyitási módoknak felel meg:

Érték	Egységszám	BASIC megfelelő
1	8	OPEN 8,11,15
2	9	OPEN 9,12,16
3	10	OPEN 10,13,17
4	11	OPEN 11,14,18

A rutin lefutása után a kívánt blokk a 0FE00H címen kezdődő pufferbe kerül, míg a VIC$DATA változó értéke az olvasás eredményéről tájékoztat:

0 ($00)	nincs hiba
11 ($0B)	a lemezt az egységben kicserélték
13 ($0D)	írás/olvasási vagy csatorna hiba
15 ($0F)	a lemezegység nincs a rendszerben

2. alfunkció: 1541-es írás. Az adatok beállítása hasonló, mint az előbbi rutinnál. Az adatokat a rutin a 0FE00H pufferből írja ki.

3. alfunkció: 1571-es előkészítése olvasásra (MFM vagy GCR formátum). A sáv és szektorszámot a VIC$TRACK, illetve VIC$SECTOR változókban kell megadni. Ha egy lemez hátoldalát akarjuk olvasni, akkor a VIC$SECTOR 7. bitjét magasra kell állítani. Ezek lehetséges értéke a használt lemez formátumától függ. A beolvasandó blokkok számát a VIC$COUNT változóban kell megadni. Ha a (FAST AND VIC$DRV) érték nullától különböző, akkor az adatátvitel a gyors protokoll szerint történik. A rutin végén a VIC$DATA jelentése ugyanaz, mint a 1541-es írás/olvasáskor. Kivétel: a 12 ($0C), aminek jelentése: a lemezegység nem 1571-es (vagy 1570-es). Ha gyorsolvasást állítottunk be és a rutin lefutása után (FAST AND VIC$DRV) nem nulla, akkor 1541-es egységről van szó.

4. alfunkció: 1571 előkészítése írásra. A sáv és szektorszámot ugyanúgy kell beállítani, mint az előző esetben. Ugyancsak meg kell adni a VIC$COUNT értéket is.

5. alfunkció: 1541-es vagy 1571-es mód lekérdezése. A használni kívánt lemezegység számát (8–11) a VIC$DRV változóba kell helyezni. A rutin ellenőrzi, hogy MFM vagy GCR formátumú lemez van-e benne, beállítja a FAST írási/olvasási módot. A rutin végén a VIC$DATA alsó 4 bitje a FAST-mód állapotkódját, felső 4 bitje pedig az MFM lemezek szektor méretét tartalmazza.

6. alfunkció: lemezegység lekérdezése. A rutin (ha hibátlan) a lemez állapotával, a szektormérettel (MFM esetén), illetve 0FE00H-tól 6 byte további információval tér vissza. Ezek a következők:

FE00	sáv állapota
FE01	szektor szám sávonként
FE02	logikai sáv
FE03	minimum szektor szám (ezen a sávon)
FE04	maximum szektor szám (ezen a sávon)
FE05	töltő karakter (physical interleave)

VIC$DATA tartalmazza az esetleges hiba okát, hasonlóan az 1. alfunkcióhoz.

7. alfunkció: karakter küldése a nyomtatóra. A rutin meghívása előtt VIC$DRV-ba kell beírni az egységszámot (4,5), VIC$TRACK-be azt a másodlagos címet, amivel a nyomtatót megnyitottuk. VIC$DATA tartalmazza a kiírandó karaktert, feltéve, hogy VIC$COUNT=0. Ha VIC$COUNT nem 0, akkor annyi darab karakter íródik ki. Az első kiírandó karakter helyére az FE00-tól levő pufferben a 0FE00H mutat.

8. alfunkció: 1541-es vagy 1571-es lemez megformázása. Ha a FAST mód megengedett, akkor egy, a 0FE01H-tól kezdődő parancsot is elküld a rendszer. A parancs hossza a 0FE00H-ban van. A parancsot mindig megelőzi egy "U0" elküldése, ezért a parancsnak ezt a részét nem szabad a pufferba beírni.

9. alfunkció: Felhasználói 8502-es kódú rutin meghívása. A rutin kezdőcímét az FD05,FD06 helyekre kell tölteni alsó, felső byte sorrendben. A 8502-es processzor bekapcsolásakor az MMU regiszter értéke $3E. Ez a 0-ás szeletet és az I/O regiszterek használatát jelenti. A visszatéréskor ennek ugyanígy kell lennie. A Z80 visszakapcsolása a RTS (8502-es kód!) végrehajtása után történik. Ekkor a 8502-est 1MHz-es órajellel kell üzemeltetni, különben a rendszer lemerevedhet.

10. alfunkció: RAM lemez olvasása.

11. alfunkció: RAM lemez írása.

12. alfunkció: a <40/80 DISPLAY> billentyű állapotának lekérdezése. A rutin az A regiszterbe olvassa az MMU $D505 regiszterének tartalmát. Ha a 7. bit magas (1), akkor a <40/80 DISPLAY> billentyű nincs lenyomva, ha a bit alacsony, akkor lenyomott állapotban van.

255. funkció: Rendszer dátum beállítása
További paraméterek: HL tartalmazza az 1978. január 1-e óta eltelt napok számát.

13. fejezet

Turbo Pascal

13.1 A Turbo Pascal(r) felépítése

A CP/M+ operációs rendszer egyik előnye, hogy a CP/M+ operációs rendszer alá készített (azaz a szabvány BIOS és BDOS funkciókat használó) Z80-as programok változtatás nélkül használhatók. A 1571-es lemezegység használatakor még az az előny is megvan, hogy ugyanazt az adathordozót használhatjuk, mint a többi CP/M gép esetében s nincs szükség bonyolult konverziókra.

A 16 bites gépeken a Turbo Pascal ma már azzal fenyeget, hogy kiszorítja a 8-bites gépeken még ugyancsak elterjedt BASIC-et, s egyeduralkodóvá válik. Ahogy a Borland International reklámja szól: a Turbo Pascal a század programozási nyelve.

A Turbo Pascal a Standard Pascal egyik nyelvjárása. File kezelése lényegesen eltér attól s számos eljárás és függvény egészíti ki. A Turbo elnevezés indokolt, a Turbo Pascal jelenleg a leggyorsabb Pascal fordító. A mérete sem megvetendő: a V2.03A verziójú fordító mindössze 30Kbyte! A 8 és 16 bites változatokból 1985 végéig több mint 400,000 példányt adtak el!

A Turbo Pascal 8 bites változatát a Borland International a Commodore 128-ra is implementálta. Ebben a fejezetben ennek a programnak a használatát ismertetjük.

A Turbo Pascal egy memória rezidens program szerkesztőből, egymenetes fordítóból és futtató rendszerből áll. Az egyszerűbb esetben a memóriában a program szövege, a lefordított program és a program adatai is elférnek. Nagyobb programok esetén lehetőség van a lefordított program különálló futtatására is. Ekkor a rendszerből csak a futtató rész (és természetesen a lefordított program) van a memóriában. Így közel 22Kbyte memória nyerhető. Végül lehetőség van átlapolt programok fordítására és futtatására. A programok szövege forrás és gépi kódú programrészekre való hivatkozásokat tartalmazhat.

A CP/M+ indítása után (lásd I.kötet 153. oldal) a Turbo Pascal rendszer a

```
A>Turbo
```

parancs kiadásával indítható. Ha gyakran használjuk a rendszert, célszerű a fenti parancsot a PROFILE.SUB file-ba is beiktatni (lásd I.kötet 168. oldal).

A Turbo Pascal programszerkesztője többkarakteres parancsokat is használ. Ezért a funkcióbillentyűket s esetleg a numerikus billentyűzet néhány billentyűjét célszerű ezekre a karakterekre átprogramozni. Erre a 11. fejezetben ismertetett KEYFIG segédprogramot használhatjuk. Az átprogramozás után a KEYFIG-gel a billentyűk új definícióit elmenthetjük a CP/M lemezre. Ezek után a rendszer betöltésekor az új billentyű definíciók automatikusan érvényessé válnak.

A Turbo Pascal esetén az alábbi billentyűzet kiosztást célszerű használni:

Billentyű	Karakter sorozat	Jelentés
<F1>	↑QB	blokk elejére
<F2>	↑QK	blokk végére
<F3>	↑KB	blokk elejének beállítása
<F4>	↑KK	blokk végének beállítása
<F5>	↑KC	blokk másolása
<F6>	↑KH	blokk kijelzésének törlése
<F7>	↑KR	blokk beolvasása
<F8>	↑KW	blokk kiírása
<STOP>	↑KD	szerkesztés vége
<ENTER>	↑KDS	szerkesztés vége és a szövegfile elmentése
<+[n]>	↑QR	szövegfile elejére
<-[n]>	↑QC	szövegfile végére

[n] : a numerikus billentyűzeten!

A rendszer betöltése után a következő logo jelenik meg:

```
TURBO Pascal system                         Version x.xxA
                                            CP/M-80,  Z80

            Copyright (c) 1983,1984 by BORLAND INC.

Terminal: Commodore 128/ADM 31


 Include error messages (Y/N)? █


 Loading A:TURBO.MSG
```

A bejelentkezés után a rendszer megkérdezi, hogy a hibaüzenetek szövegét betöltse-e a memóriába. Igen válasz, azaz az <Y> billentyű megnyomása után a szövegfile a memóriába töltődik. Ha az <N> billentyű megnyomásával válaszolunk, akkor a hibáknak csak a sorszáma íródik ki. A hibaüzenetek mintegy 2Kbyte-tal csökkentik a rendelkezésre álló memóriaterület nagyságát.

A fentiek után megjelenik a Turbo Pascal főmenüje:

```
Logged drive: A

Work file:
Main file:

Edit            Compile          Run              Save

eXecute         Dir              Quit             compiler Options


Text:      0 bytes(802A-802A)
Free: 27099 bytes(802B-EA06)


>▮
```

A menüből a zöld (fekete-fehér monitoron sötétebb színű) nagybetűvel megjelölt billentyűk lenyomásával lehet választani. A SHIFT váltót nem kell használni. (A szabad memóriaterület vége attól függ, hogy milyen rezidens bővítések vannak a rendszerben. A szöveg eleje mutató attól függ, hogy betöltöttük-e a hibaüzenetek szövegeit-e vagy sem.

Az egyes menüpontok használata a következő:

L: az aktuális lemezegység betűjelét kell megadni. Erre a lemezegységre menti a fordító a szövegfile-okat és ezen helyezi el a lefordított programot. A lehetséges érték az A-P tartományba esik.

D: az aktuális lemezegység katalógusát a képernyőre listázza. A menüpont kiválasztása után megkérdezi, hogy milyen file nevekre vagyunk kiváncsiak. A * és ? karakterek használhatók. Ha nem adunk meg paramétert, az a *.*-nak választásának felel meg.

Q: visszatérés a CP/M operációs rendszerhez. A rendszer nem kérdez vissza, hogy valóban ki akarunk-e lépni a programból. Ha azonban a 'Work' file-t még nem mentettük el, akkor rákérdez, hogy elmentse-e. Ha 'Y'-nal válaszolunk, akkor a mentés megtörténik.

C: A kiválasztott opciónak megfelelően lefordítja a programot.

O: A fordítási opció kiválasztása. A Turbo Pascal programokat lényegében véve háromféleképpen lehet lefordítani. Ennek megfelelően a következő almenü jelenik meg:

```
compile →  Memory
           Com-file
           cHn-file

Find run time error  Quit
```

Az almenüből újból a nagybetűk szerint lehet választani. A "compile →" felirat a lehetséges három fordítási mód közül az utoljára kiválasztott előtt jelenik meg. Az egyes opciók jelentése a következő:

Q: visszatérés a főmenüre.

F: Ha nem a memóriába fordított programot futtatunk, akkor futás közbeni hiba esetén a program a hibaüzenet mellett az utasításszámláló értékét írja ki. A fenti opció segítségével megkereshetjük a hiba helyét. Az opció választása után az első kérdésre az utasításszámláló értékét, majd a program nevét kell beírni. Ezután – mintha a memóriában levő program okozta volna a hibát – a szerkesztőbe kerülünk, s a kurzor a hibát okozó sorra áll.

A **C**, **H** és **M** opciók választásával a fordítás módját szabályozhatjuk. A **C** opció esetén egy .COM file-t állít elő a fordító, amit a CP/M-ből közvetlenül futtathatunk. Ha a **H** opciót választjuk, akkor a fordítás ugyancsak egy lemezes file-t állít elő, de a run-time modult a rendszer nem fordítja hozzá, s a filenév kiterjesztés .CHN lesz. Az ilyen file-okat átlapolt programok írására használhatjuk. Végül az **M** opció választása után a rendszer a memóriába fordít. Ebben az esetben a lefordított programot nem tudjuk lemezre menteni. (Ez utóbbi a fordító alapértelmezése.)

X: a Turbo Pascal lehetőséget biztosít tetszőleges .COM végű file futtatására. A szövegfile-t automatikusan elmenti, betölti a TURBO.OVL programot, majd végrehajtja a kért parancsot. Ezután a TURBO.OVL segítségével visszatölti magát a TURBO-t és az elmentett szövegfile-t.

W: A szerkesztendő file nevének megadása. A funkció választása után – ha előtte más file-t szerkesztettünk, s még nem mentettük el, a rendszer megkérdezi, hogy elmentse-e. Ezután megkisérli az aktuális lemezegységről betölteni a file-t. Ha nem találja, akkor úgy értelmezi, hogy új file-t kívánunk létrehozni.

M: A Turbo Pascal lehetőséget nyújt arra, hogy a program szövegében könyvtári file-okra hivatkozzunk. Gyakran előfordul az, hogy a programon magán nem, csak a benne használt egyik könyvtári programrészen módosítunk. Ebben az esetben a 'Main' file-nak az eredeti programot, míg a 'Work' file-nak a könyvtári programot kell megnevezni. Ha 'Main' file nevet is megadunk, akkor a szerkesztőből való kilépés után a Turbo Pascal azonnal elmenti az éppen szerkesztett file-t, betölti és lefordítja a 'Main' file-t.

E: Áttérés szerkesztő módba.

C: A memóriában tárolt program lefordítása.

R: A memóriában tárolt program lefordítása és futtatása. Ha a legutóbbi fordítás óta nem módosítottuk a szöveget, akkor a fordítás elmarad.

A Turbo Pascal programok írása lényegesen eltér attól, ahogy a Commodore 64 vagy 128-as módban kell a programokat megírni. A leglényegesebb eltérések a következők:

- a soroknak nincs sorszámuk;
- egy új karakter beírásakor a program azonnal módosul;
- a sor vége nem az utasítás vége is egyben;
- egyszerre a teljes program módosítható.

A program szövege tulajdonképpen egyetlen hosszú sornak tekintendő, amit csak technikai okok miatt tördelünk szét sorokra. Egyetlen megkötés, hogy egy sor nem lehet hosszabb 127 karakternél. A programban csak a Turbo Pascal karaktereit használhatjuk. Nincs például lehetőség a vezérlő karakterek grafikus karakterenként történő beírására, csak a chr függvényt használhatjuk! A Commodore-os programszerkesztők idézőjel üzemmódjának megfelelője sem létezik. (Összehasonlításul: I.kötet 16.oldal)

Ezen túlmenően mégis csak van egy megkötés a programok írására, ez azonban nem kötelező, hanem a struktúrák egymáshoz való viszonyának jobb elősegítését szolgálja. Az alacsonyabb rendű struktúra szövegrésze általában 3-4 karakterrel az öt magában foglaló struktúra szövegénél beljebb kezdődik. A fejezet példáiban ezt minden alkalommal megfigyelhetjük.

A Turbo Pascal szerkesztője nem teszi automatikusan lehetővé, hogy a megszerkesztett programot egy másik néven mentsük el. Az elmentett program mindig a Work file-ban megadott file-ba kerül. Ha mégis más néven akarjuk tárolni, akkor a program elejét és végét jelöljük meg (<CTRL-K>B, <CTRL-K>K) és a <CTRL-K>W vezérlő karaktersorozattal, mint egyetlen blokkot írjuk ki. A szerkesztő parancs kérni fogja a file nevét, majd elmenti a kijelölt blokkot (esetünkben az egész file-t).

A program elmentése a főmenüből a Save kiválasztásával történik. Ha azon a néven már volt egy program, akkor azt a szerkesztő átnevezi. Pontosabban, ha a 'Work file' kérdésére a PROBA.PAS file nevet adtuk meg, de ilyen file még nincs a lemezen, akkor New file üzenetet kapunk, és a program elmentésekor hozza létre a szerkesztő a file-t. Ha a file már létezik, akkor automatikusan a szerkesztő munkaterületére olvasódik. A program elmentésekor a régi file nevét a rendszer PROBA.BAK-ra változtatja, s az új programszöveget a szerkesztő a PROBA.PAS file-ba menti el.

13.2 A Turbo Pascal(r) programszerkesztője

A főmenüből az E funkció kiválasztásával léphetünk be a szerkesztőbe. Ha a fordítás közben hiba történik, a rendszer ugyancsak a szerkesztőbe tér vissza. A szerkesztőből kilépni a ↑KD (<CTRL-K><D>) billentyűzéssel lehet.

Szerkesztéskor a képernyő első sora az ún. **parancssor**:

```
Line 1        Col 1       Insert        Indent      A:filenév
```

A parancssor fehér betűkből áll, míg a rendes szöveg ettől eltérő színű (zöld vagy sárga) betűket tartalmaz. Ha kijelölünk egy blokkot, az ugyancsak ilyen színűre változik.

Szerkesztésre a teljes képernyőt használhatjuk. Ha a program teljes szövege nem fér rá a képernyőre, akkor a képernyőt ablakként mozgathatjuk a szöveg felett. A kurzor mindenkori helyét a parancssorról olvashatjuk le. A parancssor jelzi azt is, hogy az éppen beírt karakter beszúródik-e a sorba (Insert) vagy felülírja a kurzor helyén levő karaktert (Overwrite).

A kurzor mozgatására az alábbi karakterek használhatók:

Funkció	Vezérlő karakter	Egyéb
Kurzor egy karakterrel balra	<CTRL-S>	<BALRA>
Kurzor egy karakterrel jobbra	<CTRL-D>	<JOBBRA>
Kurzor a következő tabulálási pontra	<CTRL-I>	<TAB>
Kurzor egy sorral feljebb	<CTRL-E>	<FEL>
Kurzor egy sorral lejjebb	<CTRL-X>	<LE>
Kurzor a következő szóra	<CTRL-F>	<CRSR BALRA>
Kurzor az előző szóra	<CTRL-A>	<CRSR JOBBRA>
Képernyő feltolása egy sorral	<CTRL-W>	<CRSR FEL>
Képernyő letolása egy sorral	<CTRL-Z>	<CRSR LE>
Képernyő feltolása egy lappal	<CTRL-R>	
Képernyő letolása egy lappal	<CTRL-C>	
Új sor kezdés	<CTRL-M>	<RETURN>

A TAB pozíciók az előző sorban álló szavak első pozíciójára állítódnak be. Ha az IDENT paramétert bekapcsoltuk, akkor a <RETURN> hatására még egy TAB karakter is kiíródik, s a kurzor az előző sor első nem szóköz karaktere alá kerül. Ha a sor üres volt, akkor a következő sor elejére.

A fenti egyszerű kurzor mozgásokat az alábbiak egészítik ki:

Funkció	Vezérlő karakter
Sor elejére	<CTRL-Q>S
Sor végére	<CTRL-Q>D
Képernyő tetejére(2.sor)	<CTRL-Q>E
Képernyő aljára(24.sor)	<CTRL-Q>X
Szöveg elejére	<CTRL-Q>R
Szöveg végére	<CTRL-Q>C
A kurzor utolsó helyére	<CTRL-Q>P

A szerkesztő egy sor törlő és beszúró utasítást tartalmaz. Ezek segítségével karakterek, szavak és teljes sorok törölhetők. Lehetőség van kontrol karaterek beillesztésére. Ezek a kiírás formátumát vezérlik, a programot nem befolyásolják. A törlő/beszúró vezérlő karakterek az alábbiak:

Funkció	Vezérlő karakter
A kurzor helyén levő karakter törlése	<CTRL-G>
A kurzor utáni szó törlése	<CTRL-T>
A kurzor sorának törlése	<CTRL-Y>
A kurzortól a sor végéig törlés	<CTRL-Q>Y
Új sor beszúrása	<CTRL-N>
Kontrol karakter beszúrása	<CTRL-P><kontrol kar.>

A Turbo Pascal egy olyan lehetőséget nyújt a programíráshoz, amihez hasonló a BASIC-ben nincs meg: lehetőség van **adott szövegrész megkeresésére és egy másikra való cserélésére**. A keresés és csere **módját** szabályozni lehet. Ezek a módok a következők lehetnek:

B: a keresés és a csere a kurzor helyétől **visszafelé** történik.

G: a keresés és csere – a kurzor helyétől a **teljes szövegben** történik.

xxx: xxx előjel nélküli egész szám. A keresés (és azután a csere) a megadott szöveg xxx-edik megjelenéséig tart.

U: a keresés során a nagy és kisbetűkből adódó eltéréseket a szerkesztő figyelmen kívül hagyja. Pl. ha a KATI-t keressük, akkor a szerkesztő a Kati-t is megtalálja.

W: Csak egész szavakat talál meg. A mód megadásakor a szerkesztő a KATIKA-t nem jelzi, ha a KATI-t keressük.

N: Csere esetén nem kérdez vissza. Ha nem választjuk ezt a módot, akkor minden csere előtt visszakérdez, hogy végrehajtsa-e.

A keresés vagy csere végrehajtásához ki kell adni a megfelelő vezérlő karaktersorozatot, beírni a keresendő és cserélendő karaktersorozatot, megadni a módot. A rendszer ezután kezdi csak el a keresést, illetve a cserét. A használható karaktersorozatok az alábbiak:

Funkció	Módok		Vezérlő karakter
Karaktersorozat keresése	BGUW	xxx	<CTRL-Q>F
Karaktersorozat cseréje	BGUWN	xxx	<CTRL-Q>A
Az utolsó cserélő vagy kereső parancs ismétlése			<CTRL-L>

További vezérlő karakterek lehetővé teszik a szöveg egy tetszőleges részének kijelölését. Ezt a részt blokknak hívjuk. A blokkot a blokk elejének és végének kijelölésével lehet megadni. A blokkban levő szöveg fehér betűkkel jelenik meg a képernyőn. A kijelzés ilyen módja megszüntethető. Ezzel a blokk nem szünik meg, csak nem látszik. A maradék vezérlő karakterek a következők:

Funkció	Vezérlő karakter
Blokk elejének megjelölése	<CTRL-K>B
Blokk végének megjelölése	<CTRL-K>K
Kurzor a blokk elejére	<CTRL-Q>B
Kurzor a blokk végére	<CTRL-Q>K
Egyetlen szóból álló blokk megjelölése	<CTRL-K>T
Blokk törlése	<CTRL-K>Y
Blokk kijelzésének törlése (fehér)	<CTRL-K>H
Blokk kiírása lemezes file-ba	<CTRL-K>W
Blokk áthelyezése a kurzor helyére	<CTRL-K>V
Blokk beolvasása lemezes file-ból	<CTRL-K>R
Beszúrási/felülírási mód váltás	<CTRL-V>
Automatikus bekezdés ki/be kapcsolása	<CTRL-Q>I
A kiadott parancs törlése	<CTRL-U>
A sor előző tartalmának visszaállítása	<CTRL-Q>L
Visszatérés a főmenühöz	<CTRL-K>D

A kiadott parancs törlése természetesen a parancs már végrehajtott részének hatását nem szünteti meg. A <CTRL-Q>L parancs működési mechanizmusa a következő. Ha a kurzorral egy új sorra lépünk, akkor annak tartalmát a szerkesztő elmenti. Ez az eredeti sor íratható vissza ezzel a paranccsal, amíg a sort el nem hagytuk. Ha a sort töröljük a <CTRL-Y>, akkor ezzel az utasítással nem tudjuk a törölt sort visszaállítani.

A Turbo Pascal programok írása lényegesen eltér attól, ahogy a Commodore 64 vagy 128-as módban kell a programokat megírni. A leglényegesebb eltérések a következők:

- a soroknak nincs sorszámuk;
- egy új karakter beírásakor a program azonnal módosul;
- a sor vége nem az utasítás vége is egyben;
- egyszerre a teljes program módosítható.

A program szövege tulajdonképpen egyetlen hosszú sornak tekintendő, amit csak technikai okok miatt tördelünk szét sorokra. Egyetlen megkötés, hogy egy sor nem lehet hosszabb 127 karakternél. A programban csak a Turbo Pascal karaktereit használhatjuk. Nincs például lehetőség a vezérlő karakterek grafikus karakterenként történő beírására, csak a chr függvényt használhatjuk! A Commodore-os programszerkesztők idézőjel üzemmódjának megfelelője sem létezik. (Összehasonlításul: I.kötet 16.oldal)

Ezen túlmenően mégis csak van egy megkötés a programok írására, ez azonban nem kötelező, hanem a struktúrák egymáshoz való viszonyának jobb elősegítését szolgálja. Az alacsonyabb rendű struktúra szövegrésze általában **3-4 karakterrel** az öt magában foglaló struktúra szövegénél **beljebb** kezdődik. A fejezet példáiban ezt minden alkalommal megfigyelhetjük.

A Turbo Pascal szerkesztője nem teszi automatikusan lehetővé, hogy a megszerkesztett programot egy másik néven mentsük el. Az elmentett program mindig a **Work file**-ban megadott file-ba kerül. Ha mégis más néven akarjuk tárolni, akkor a program elejét és végét jelöljük meg (<CTRL-K>B, <CTRL-K>K) és a <CTRL-K>W vezérlő karaktersorozattal, mint egyetlen blokkot írjuk ki. A szerkesztő parancs kérni fogja a file nevét, majd elmenti a kijelölt blokkot (esetünkben az egész file-t).

A program elmentése a főmenüből a **S**ave kiválasztásával történik. Ha azon a néven már volt egy program, akkor azt a szerkesztő átnevezi. Pontosabban, ha a 'Work file' kérdésére a PROBA.PAS file nevet adtuk meg, de ilyen file még nincs a lemezen, akkor **New file** üzenetet kapunk, és a program elmentésekor hozza létre a szerkesztő a file-t. Ha a file már létezik, akkor automatikusan a szerkesztő munkaterületére olvasódik. A program elmentésekor a régi file nevét a rendszer PROBA.BAK-ra változtatja, s az új programszöveget a szerkesztő a PROBA.PAS file-ba menti el.

13.3 A nyelv alapjai

A Turbo Pascal - szemben a BASIC-kel vagy a FORTRAN-nal - struktúrált programozási nyelv. Ez egyes programstruktúrák vagy teljes egészében tartalmazzák egymást, vagy nincs közös részük. Egymásba skatulyázott programstruktúrák esetén a belső használhatja a külső változóit, de fordítva nem.

A Turbo Pascal további sajátja, hogy **típusos** nyelv, azaz minden egyes használt változó típusát - annak használata előtt - meg kell adni. Hasonló igaz az eljárásokra és függvényekre is.

A Turbo Pascal programok az alábbi karaktereket használhatják:
- a kis- és nagybetűk: a-z és A-Z
- a számjegyek: 0-9
- az aláhúzás jele: _
- speciális jelek: + - * / = < > ( ) [ ] { } . , : ' # $ ↑ és a szóköz

Szemben a BASIC-kel a speciális jelek mindegyike elválasztó jelnek is számít. A szerkesztő a **szó** fogalmát erre alapozza. Ha tehát a <CTRL-A> parancsot adjuk ki, akkor a kurzor visszafelé mozog az első elválasztó jelig, vagy szóközig. A szóköz kivételével a fenti speciális jeleknek saját jelentése van, s adott helyen nem cserélhetők fel egymással.

A fordító **nem tesz különbséget a kis- és nagybetűk között**. Ha tehát azt írjuk, hogy sajatprogram vagy SajatProgam, az ugyanannak felel meg. Célszerű olyan neveket használni, ami pontosan leírja, hogy mire is használjuk. A név hossza maximálisan 127 karakterből állhat. Az aláhúzás jelet is a nevek olvashatóbbá tétele érdekében szokás használni. Pl. célszerűbb a Kezdo_Ertek írásmód, mind a kezdoertek használata. Mindez természetesen csak a programok olvashatóságát növeli.

A Turbo Pascal 43 ún. **alapszót** ismer. Ezeket más célra nem lehet felhasználni. Ezeken túl több mint 100 előre definiált szót tartalmaz a nyelv. Ezek újra definiálhatók. (Bár ezt a gyakorlatot nem javasoljuk!) Az alapszavak a következők:

absolute	**and**	**array**	**begin**
case	**const**	**div**	**do**
downto	**else**	**end**	**external**
file	**for**	**forward**	**function**
goto	**if**	**in**	**inline**
label	**mod**	**nil**	**not**
of	**or**	**packed**	**procedure**
program	**record**	**repeat**	**set**
shl	**shr**	**string**	**then**
to	**type**	**until**	**var**
while	**with**	**xor**	

A programban használt nevek részként tartalmazhatják ezeket a szavakat. Pl. a new_file elnevezés használható, annak ellenére, hogy a file egy alapszó, a new pedig egy előre definiált szó. Egy ilyen azonosító használata sem a file, sem a new jelentését nem befolyásolja. Ennek oka, hogy a fordító az elválasztó jelek közti részt tekinti egyetlen logikai egységnek. (A Commodore BASIC fordítók az alapszó végét - esetünkben a new-nál - tekintik elválasztójelnek.)

Azonosítók

A Turbo Pascal azonosítói (vagy a használt nevek) betűvel vagy aláhúzással kezdődnek s ezt betűk, számjegyek és aláhúzások tetszőleges kombinációja követheti. Az azonosító hosszának csak az szab határt hogy a szerkesztő egyetlen sorában el kell férnie: ez 127 karakter. 15–20 karakternél hosszabb azonosítók használata már rettentően kényelmetlen. (Lehet azt a gyakorlatot is követni, hogy 2-3 karakterből álló neveket használunk, majd a program elkészítése után a <CTRL-Q>A szerkesztő paranccsal hosszabb, de kifejezőbb nevekre cseréljük.)

Programstruktúra

A Turbo Pascal (s a Standard Pascal) programok is az alábbi formában épülnek fel:

<program fejléc>
<deklarációs rész>
<végrehajtható rész>.

Nézzük sorra, hogy az egyes részek miből is állnak!

A <program fejléc> – szemben a Standard Pascal-lal el is maradhat. Ha mégis szerepel, akkor a **program** alapszóval kezdődik, amit a program neve és esetleg zárójelben vesszővel elválasztva a programban használt file-ok nevei követnek. Mindezt egy pontosvessző zárja le. A <program fejléc>-et a fordító semmire sem használja, csak azért adható meg, hogy a Standard Pascallal a kompatibilitás megmaradjon. Azt javasoljuk, hogy a <program fejléc>-et

program <program neve>;

alakban használják.

A <deklarációs rész>-ben mindazokat az objektumokat (program komponenseket) deklarálni kell, amiket a <végrehajtható rész>-ben használni akarunk. A <deklarációs rész> címke, konstans, típus, változó, eljárás és függvény deklarációkat tartalmazhat. A Standard Pascal-ban a fenti felsorolás egyben kötelező sorrend is, a Turbo-ban azonban nincs ilyen kikötés. Ne felejtsük el azonban, hogy mielőtt egy objektumot használni akarunk, előtte definiálni kell. Ha tehát a mondat_kiírasa eljárásban használni akarjuk a szó_kiírasa eljárást, akkor először a szó_kiírasa eljárást kell deklarálnunk, majd a mondat_kiírasa eljárást.

A <végrehajtható rész> tartalmazza azokat az utasításokat, amelyeket a programnak végre kell hajtania. Vegyük észre a program végén álló **.**-t! Minden programnak ezzel kell végződnie.

Pascal deklarációk

Címke deklarációk

Bár a Turbo Pascal struktúrált nyelv, tervezői mégis lehetővé tették a "**goto** <címke>" alakú vezérlésátadást. Azokat az azonosítókat, amelyeket címkeként szeretnénk használni, a címke deklarációs részben kell felsorolni. Ennek alakja

label <címke lista>;

ahol a címke lista egymástól vesszővel (,) elválasztott címkéket tartalmaz. A deklarációt egy további pontosvessző zárja.

Konstans deklarációk

A konstansok olyan változók, amelyek értékét nem lehet megváltoztatni. A konstans deklaráció megadja a konstans típusát és értékét is. A konstans definíció alakja

<konstans azonosító> = { <literál> | <előjel><konstans azonosító> }

A konstansok definíciója miatt meg kell különböztetnünk magát a konstans azonosítóját, s az értéke megadására szolgáló jelsorozatot. Ez utóbbit hívjuk <literál>-nak. A <literál> előjel nélküli vagy előjeles szám, illetve karaktersorozat lehet. A szimbolikus konstansok használata növeli a programok olvashatóságát.

A konstans deklaráció alakja a következő:

const <konstans definíció lista>;

ahol a <konstans definíció lista> egymástól vesszővel (,) elválasztott konstans definíciókat tartalmaz.

Típus deklarációk

A Turbo Pascalban hat előre definiált típust használhatunk, ezek a REAL, INTEGER, BOOLEAN, CHAR, BYTE, STRING típusok. További lehetőségek vannak a már meglevő típusokból újak előállítására. Ez a lehetőség az egyike azoknak, amelyek a PASCAL-t sok egyéb programozási nyelvtől megkülönböztetik. Új típusok előállítására az ún. **típus-műveletek** szolgálnak. Egyes típus-műveletek egyben azt is meghatározzák, hogy az általuk létrehozott új típuson milyen műveletek végezhetők.

A típusdeklarációs általános alakja a következő:

<típus azonosító> = <típus leírás>

ahol a <típus leírás> a típus-műveletek és az előre definiált típusok segítségével épül fel. A típus deklarációs rész - ha egyáltalán szerepel - így néz ki:

type <típus definíció lista>;

ahol a <típus definíció lista> egymástól ;-vel elválasztott típus definíciókat tartalmaz. A típus definícióknál is érvényes az az elv, hogy egy típus definíciójában csak már definiált típusok használhatók fel.

Változó deklarációk

A típus-deklarációk segítségével megadhatjuk, hogy milyen adatokkal is akarunk dolgozni. A változó adott típusú adatok tárolására szolgál. A változó definiálásakor meg kell mondani, hogy milyen típusú. A változó definíció lehetőséget biztosít egyidejűleg több azonos típusú változó definiálására is. Egy változó definíció az alábbi formájú:

<változó lista>:<típus leírás>

A <változó lista> egymástól vesszővel (,) elválasztott neveket tartalmaz. Minden egyes név típusa a <típus leírás>-nak megfelelő típusú lesz. A <típus> vagy előre definiált, vagy ezt megelőzően már típusdeklarációval megadott típus kell hogy legyen.

A változó deklarációs rész általános alakja a következő:

var <típus definíció lista>;

A <változó definició lista egymástól pontosvesszővel (;) elválasztott változó definíciókat tartalmaz.

A továbbiakban felsoroljuk az előre definiált típusokat, s a rajtuk értelmezett műveleteket.

INTEGER
Az egész számok használatára a Turbo Pascal az INTEGER előre definiált típust biztosítja. Az INTEGER típusú számoknak a -32767..32768 intervallumba kell esniük. Ennek megfelelően a fordító ezeket a számokat 16 bites előjeles számként tárolja. A rendszer tartalmaz egy MAXINT nevű INTEGER változót, aminek az értéke 32768. Az INTEGER típus egy ún. **rendezett** típus (erről a későbbiekben részletesen lesz szó). A rendezés a természetes számokon szokásos rendezés.

Egész számokat előjeles és előjel nélküli formában adhatunk meg. Használhatjuk még a $ jelölést is, ekkor a fordító egy hexadecimális számot vár. Pl. $8000=-32768, $FFFF=-1 és $7FFF=32768.

A Turbo Pascal-ban - a karakterkonverziók támogatására - előredefiniált típusként szerepel a BYTE. A BYTE típus rendezett típus és az INTEGER résztípusa. A BYTE a 0..255 intervallumba eső egész számokból áll.

REAL
A REAL előre definiált típus a valós számok használatát biztosítja. A REAL típusú változókkal a szokásos műveletek végezhetők el. A Turbo Pascal-ban a valós számok megközelítőleg az 1.0E-38 .. 1.0E+38 intervallumba esnek, pontosságuk 11 decimális jegy. A REAL típusú változók tárolására a rendszer 6 byte-ot használ.

REAL típusú literálok vagy a tizedespontot vagy az E exponenst kell hogy tartalmazzák: 123. vagy 123E0!

A Turbo Pascal-ból hiányzik az a beépített mechanizmus, ami automatikusan konvertál az egész és a valós számok között. Ha egy eljárás valahol egész számot vár, s mi valóst adunk meg, akkor hibajelzést kapunk.

Egyetlen előredefiniált REAL típusú konstans van, ez a PI. Értéke 3.1415926536.

CHAR
A Turbo Pascal a CHAR előredefiniált típust egyetlen karakterből álló sztring kezelésére használja. A CHAR használata abból adódik, hogy a rendszer megkülönbözteti a BYTE és a CHAR típust. Ezek a chr függvény segítségével kölcsönösen és egyértelműen megfelelnek egymásnak, de adott eljárások és függvények esetében vagy csak az egyik vagy csak a másik típust használhatjuk csak.

STRING
A STRING előre definiált típus, a BASIC string típusához nagy mértékben hasonlít. Az egyetlen lényeges különbség, hogy sztring változók megadásakor meg kell adni annak maximális hosszát is:

```
var
     proba: string[45];
```

A fenti deklaráció hatására a fordító 46 byte-ot foglal le. Az első (vagy 0.) byte a sztring hosszát, a többi byte a sztring karaktereinek kódját tartalmazza. Szemben a Commodore BASIC-kel a sztringek tárolása nem dinamikus. A sztringek maximális hossza 255 lehet.

STRING (vagy CHAR) literálokat idézőjelek(') közé kell írni. Ha az idézőjelek közé egyetlen karaktert írunk, akkor CHAR típusú literált kapunk. A karaktersorozatba vezérlő karaktereket is elhelyezhetünk. Erre két lehetőség van. Ha CHAR típusú literált akarunk megadni, akkor pl. a <CTRL-C> megadására a ↑C jelsorozatot kell használni. Ugyanezt a #03 vagy #$03 jelsorozattal is megadhatjuk. Ha ezt a kontroll karaktert egy STRING-ben akarjuk elhelyezni, akkor az idézőjelet be kell zárni, s szóköz nélkül az előző sorozatot beírni. Ezután (szóköz nélkül!) új idézőjelet nyithatunk:

```
const.
     Harang = ↑g;
     MasikHarang = #07;
     Esc = #$1F;
     Uzenet_Eleje = 'Uzenet'#07#07'#  ';
```

BOOLEAN
A BOOLEAN előre definiált típus mindössze két értéket tartalmaz. Ezekhez előre definiált konstansok tartoznak: **TRUE** és **FALSE**. A szokásos logikai műveletek elvégezhetők. Összehasonlítás eredményeként ugyancsak BOOLEAN értéket kapunk. A fordító a BOOLEAN változókat 1 byte-on tárolja. A BOOLEAN típus sem a BYTE sem az INTEGER típussal nem kompatibilis.

Skalár (vagy rendezett típusok)
A Turbo Pascal-ban **rendezett típusnak** számít az INTEGER, a CHAR, a BYTE és a BOOLEAN típus, míg skalárnak valamennyi rendezett típus és a REAL típus.

Lehetőség van rendezett típusok deklarálására is. Ezt olyan formában tehetjük meg, hogy felsoroljuk a típus valamennyi elemét. A felsorolás egyben sorrendjüket is megadja. Rendezett típust az alábbi formában kell definiálni:

<típus azonosító> = (<azonosító lista>);

Az azonosítólista a típus elemeinek neveit tartalmazza vesszővel elválasztva. A hét napjait pl. a következőképpen definiálhatjuk:

```
type
      HetNapjai = (Hetfo,Kedd,Szerda,Csutortok,Pentek,Szombat,Vasarnap);
```

A Turbo Pascal valamennyi rendezett típuson megengedi a rákövetkező, a megelőző és a hányadik függvények használatát:

Succ(x)	az x-et követő elem a típusban;
Pred(x)	az x-et megelőző elem a típusban;
Ord(x)	x sorszáma az adott típusba.

A fenti típusdeklarációnál maradva: Succ(Hetfo)=Kedd, Pred(Szombat)=Pentek és Ord(Szerda)=2. A skalár típus nevét egyben függvénynek is használhatjuk. HetNapjai(4) a HetNapjai típus 4. sorszámú elemét adja, ami Péntek. (Az elemek sorszámozása 0-val kezdődik!)

Résztípus

A rendezett típusok intervallumai segítségével új típust lehet deklarálni. Ehhez az intervallum két kezdőpontját kell csak megadni:

<eleje>..<vége>

A két pont (..) a definícióban a 'tól-ig'-ot reprezentálja. Például a fenti HetNapjai részeként definiálhatjuk a MunkaNapok és a MunkaszunetiNapok típusokat:

```
type
      HetNapjai = (Hetfo,Kedd,Szerda,Csutortok,Pentek,Szombat,Vasarnap);
      MunkaNapok = Hetfo..Pentek;
      MunkaszunetiNapok = Szombat..Vasárnap;
```

A résztípuson az összes olyan művelet elvégezhető, ami az eredeti típuson. A lefordított program minden egyes művelet elvégzése után ellenőrzi, hogy az eredmény is a résztípusba esik. Ha nem, akkor hibajelzést kapunk.

Műveletek

A következőkben felsoroljuk a skalár, rendezett típusokon végezhető, előre definiált műveleteket. A Turbo Pascal a műveletek végrehajtása során hat precedencia szintet különböztet meg. Ezek a következők:

1. Zárójelek: (és)
2. Ellentett képzés: -
3. **not**
4. * / **div mod and shl shr**
5. + - **or xor**
6. összehasonlítás: = **<> < > <= >= in**

Az azonos rangú műveleteket a rendszer balról jobbra értékeli ki. Vigyázat: a BASIC-ben az összehasonlítás kiértékelésének precedenciája más!

Egész típusú műveletek

+	összeadás;
−	kivonás vagy ellentett képzés;
*	szorzás;
/	osztás: az operandusok egészek vagy valósak lehetnek, az eredmény valós;
div	egész osztás: az operandusok csak egészek lehetnek, az eredmény a hányados egész része;
mod	osztási maradék: az operandusok csak egész számok lehetnek, az eredmény az osztás maradéka, ugyancsak egész szám.

Valós típusú műveletek

A valós számokon a négy alapművelet végezhető el, **nincs külön hatványozás** (x↑y):

+	összeadás;
−	kivonás vagy ellentett képzés;
*	szorzás;
/	osztás.

Bitműveletek

INTEGER vagy BYTE típuson bitenkénti logikai műveleteket lehet végezni. Ezek a következők:

not	bitenkénti invertálás;
and	bitenkénti logikai és;
or	bitenkénti logikai vagy;
xor	bitenkénti logikai kizáró vagy;
shr	jobbra tolás adott számú bittel, az ellenkező oldalon 0-ák lépnek be.
shl	balra tolás adott számú bittel, az ellenkező oldalon 0-ák lépnek be.

Logikai műveletek

A logikai műveleteket BOOLEAN típusú mennyiségeken lehet elvégezni. A műveletek megegyeznek a fentiekkel, de nem bitenként hatnak, hanem a BOOLEAN által reprezentált logikai értékeken:

not	logikai tagadás;
and	logikai és;
or	logikai vagy;
xor	logikai kizáró vagy;

Összehasonlítás eredményeként BOOLEAN értéket kapunk. A relációs jelek a következőket jelentik:

=	egyenlő (tetszőleges típus);
<>	nem egyenlő (tetszőleges típus);
>	nagyobb (skalár típus);
<	kisebb (skalár típus);
>=	nagyobb vagy egyenlő (skalár típus);
<=	kisebb vagy egyenlő (skalár típus).

Sztringműveletek

A BASIC-hez hasonlóan a sztringeken egyetlen művelet végezhető: ez a sztringek összefűzése. Még a jele is ugyanaz: +.

Ugyancsak a BASIC-hez hasonlóan további függvények vannak, amelyek a sztringekkel való műveletvégzést megkönnyítik. A táblázatban levő f jelenti a függvényt, e az eljárást:

Eljárás/függvény neve		Paraméterek	Leírás
Concat	f	<sztringlista>	a sztringlistában szereplő sztringek összefűzése
Copy	f	<sztring><honnan><hány>	a <sztring>-ből a <honnan> pozíciótól kezdve kivesz <hány> karaktert, s ebből egy új sztringet képez;
Length	f	<sztring>	a <sztring> hosszával tér vissza;
Pos	f	<sztring1><sztring2>	a <sztring1> <sztring2>-beli pozíciójával tér vissza; ha ilyen nincs, akkor 0-val
UpCase	f	<karakter>	a <karakter>-nek megfelelő nagybetűvel tér vissza, ha az a "a"-"z" intervallumba esik;
Str	e	<szám><sztring>	az eljárás az INTEGER vagy REAL típusú <szám> alakját <sztring>-be helyezi;
Val	e	<sztring><szám><kód>	az eljárás a <sztring>-et számmá alakítja, s ezt a <szám>-ba helyezi el. <kód> az első fel nem dolgozható karakter sorszáma. Ha <kód>=0, akkor rendben.

Függvények

A fenti függvények és műveletek a nyelv átdefiniálható részei. A Turbo Pascal még az alábbi egyváltozós függvényeket tartalmazza:

Függvény	Leírás	Argumentum	Eredmény
Abs	abszolút érték	egész	egész
Abs	abszolút érték	valós	valós
ArcTan	arkusz tangens	valós	valós
Cos	koszinusz	valós	valós
Exp	exponenciális	valós	valós
Frac	tört rész	valós	valós
Int	egész rész	valós	valós
Ln	természetes alapú logaritmus	valós	valós
Sin	szinusz	valós	valós
Sqr	négyzetre emelés	egész	egész
Sqr	négyzetre emelés	valós	valós
Sqrt	négyzetgyökvonás	valós	valós

Függvény	Leírás	Argumentum	Eredmény
Pred	előző elem	rendezett	rendezett
Succ	következő elem	rendezett	rendezett
Odd	páratlan-e	egész	logikai
Chr	ASCII konverzió	egész	CHAR
Ord	karakter konverzió	CHAR	egész
Round	kerekítés	valós	egész
Trunc	egész rész képzés	valós	egész

A BASIC-et használók figyelmébe: az sqr a Turbo-ban négyzetre emelés! A négyzetgyökvonás az sqrt! A round függvény a legközelebbi egészre kerekít. Az int nem változtatja meg a szám típusát, az továbbra is valós marad. Valós-->egész konverzióra a trunc függvényt kell használni.

A Turbo-ban nincs exponenciális függvény. Ezért az x↑y kifejezést az exp(y*ln(x)) kifejezéssel kell helyettesíteni! A szögek értéke minden esetben radiánban értendő.

13.4 Eljárások és függvények

A Turbo Pascal egyik nagy előnye a BASIC-kel (s általában az interpretiv nyelvekkel) szemben, hogy lehetőséget ad saját eljárások és függvények definiálására, amelyek a program meghatározott részében használhatók, hasonlóan a nyelvben eleve definiált eljárásokhoz és függvényekhez. A C-128 BASIC mindössze egyváltozós függvények definiálását teszi lehetővé, s annak használatakor a megkülönböztető FN jelzést is használni kell.

A használni kívánt eljárásokat és függvényeket definiálni (deklarálni) kell. Az eljárás és függvénydeklarációk alakja eltér egymástól. Az eljárásokat az alábbi formában kell megadni:

procedure <eljárás név>(<formális paraméter lista>);
<deklarációs rész>
<végrehajtható rész>;

A függvények definíciója ettől abban különbözik, hogy a függvény típusát is meg kell adni:

function <függvény név>(<formális paraméter lista>):<típus>;
<deklarációs rész>
<végrehajtható rész>;

A <formális paraméter lista> egymástól ;-vel elválasztott paraméter elemekből áll. Ha üres, a kerek zárójelekkel együtt el kell hagyni. Egy-egy ilyen elem alakja:

{**var**}<változó lista>:<típus-leírás>

ahol a változólista egymástól ,-vel elválasztott azonosítókat tartalmaz, ezek az eljárás, illetve a függvény ún. **formális paraméterei**. Ha a paraméter elem előtt a **var** szerepel, akkor a paramétereket nem érték, hanem **név szerint adjuk át az eljárásnak (függvénynek)**. Ha a var nem szerepel, akkor érték szerint. A <típus-leírás> előre definiált típus vagy a programban már definiált típus.

Az érték szerinti paraméter átadáskor a program az aktuális paraméter értékeit kiszámítja, majd átmásolja a formális paraméterekbe. Ha pl. egy 100x100-as mátrixot adunk át, ez 10000 értékadásnak felel meg. Ha név szerinti paraméter átadást alkalmazunk, a program nem hajt végre értékadást, hanem csak megmondja az eljárásnak, hol található a szóban forgó érték, s a továbbiakban az azon a helyen levő értékkel számol. A fenti esetben tehát a mátrix átmásolása elmarad. A név szerinti paraméter átadás a program helyfoglalását is csökkenti, hiszen a 100x100-as mátrix helyét nem kell az eljárásban is lefoglalni.

A függvényeket és eljárásokat egymástól eltérően kell használni. Mindkettőt a <végrehajtható rész>-ben kell meghívni. Az eljárás meghívása egy utasítás végrehajtásának felel meg, míg a függvény meghívását tetszőleges kifejezésben kell használni, hasonlóan a beépített függvényekhez. Az eljárás illetve függvényhivatkozás alakja a következő:

<eljárás név>(<aktuális paraméter lista>)
<függvény név>(<aktuális paraméter lista>)

Az <aktuális paraméter lista> egymástól vesszővel (,) elválasztott kifejezéseket tartalmaz, amelyek száma és típusa meg kell, hogy egyezzen az eljárás vagy függvény definíciójában szereplő formális paraméterekkel. Ha valamelyik formális paraméter név szerinti hivatkozást tartalmaz, akkor az annak megfelelő aktuális paraméter csak (ugyanolyan típusú) változó lehet.

13.5 Típusfüggvények

A Pascal nyelv egyik leghatékonyabb eszköze, hogy az ún. típusfüggvényekkel vagy típusműveletekkel további típusokat adhatunk meg. Ebben a részben röviden összefoglaljuk a Turbo Pascal típusfüggvényeit. Sorrendben a következőket tárgyaljuk:

array
record
set
↑
file

array

A Turbo Pascal - hasonlóan a legtöbb nyelvhez - lehetővé teszi tömbök kezelését. Eltérően azonban a legtöbbtől (különösen a FORTRAN-tól) **a tömbök indexei tetszőleges rendezett típus elemei lehetnek.** A tömböket használatuk előtt definiálni kell. A tömbdefiníció alakja a következő:

array[<index típus lista>**] of** <típus>

Az <index lista> egymástól vesszővel (,) elválasztott rendezett típusokat tartalmaz. Ezek vagy előre definiált típusok, vagy a programban már definiált típusok vagy ilyenek résztípusai. Ez utóbbi esetben annak rendezett típusnak, vagy egy rendezett típus résztípusnak kell lennie. A fenti formában szereplő <típus>-t a tömb típusának is nevezzük.

Az **array** típusfüggvényt mind típus, mind változó definíciójában lehet használni. Ennek megfelelően az alábbi definíciók megegyeznek a BASIC DIM A%(10,10) utasítás hatásával:

a:

```
type tomb=array[0..10,0..10] of INTEGER;
var A:tomb;
```

b:

```
var A:array[0..10,0..10] of INTEGER;
```

c:

```
type
    index=0..10;
    tomb=array[index,index] of INTEGER;
var A:tomb;
```

A tömbök elemeire az indexek megadásával lehet hivatkozni. A hivatkozás alakja

<tömb név>[<index lista>]

ahol az indexek számban és típusban meg kell hogy egyezzenek a <tömb név> deklarációjában szereplő rendezett típusokkal. A tömbök elemeire való hivatkozás bárhol

szerepelhat, ahol a tömb típusával azonos változó szerepelhet. Ilyen formán a tömb elemeinek külön-külön is lehet értéket adni. Az A[2,1] elemre A[2][1] alakban is hivatkozhatunk.

A másik legfontosabb adattípus a rekord. Hasonló a tömbhöz, azzal a lényeges eltéréssel, hogy a rekord elemei nem kell hogy azonos típusúak legyenek, s nem index, hanem név szerint kell a rekord elemeire hivatkozni. További eltérés, hogy lehetőség van ún. feltételes rekordok létrehozására is. A rekord típusfüggvény alakja:

rekord <mezőlista> **end**

A <mezőlista> egymástól ;-vel elválasztott <mező definíció>-kat tartalmaz. Az egyes mezők vagy rögzített vagy feltételes mezők lehetnek. A rögzített mezők esetében a mező típusa mindig adott, míg a feltételes mező esetén egy rendezett típusú változó értékétől függ. Ennek megfelelően más formában kell definiálni a rögzített, illetve a feltételes mezőket.

A rögzített mezőket

<mezőnév lista>:<típus>

alakban kell megadni. A definíció a <mezőnév listá>-ban szereplő mezőket <típus> típusúnak definiálja. A mezők sorrendje a <mezőnév listá>-ban szereplő sorrenddel azonos. Ezeket a neveket semmilyen más célra nem használhatjuk. A <mezőnév lista> elemeit vesszővel kell elválasztani.

A feltételes mező definíció alakja a következő:

case <feltétel>:<típus> **of** <változó mezőnév lista>

A <feltétel> tetszőleges változó. Ennek értékétől függ, hogy a rekord definíciójának melyik része lesz érvényes. <típus> a szóbanforgó változó típusa, ami csak rendezett típus lehet. A <változó mezőnév lista> - szemben a rögzített mezőnevekkel, nemcsak a mező nevét és típusát, hanem azokat az értékeket is tartalmazza, amelyek esetén a rekordnak ez a része érvényes. Ennek megfelelően a <változó mezőnév lista> egymástól ;-vel elválasztott <változó mezőnev>-eket tartalmaz, amelyek alakja:
<feltétel lista>:(<mezőlista>)

A <feltétel lista> a <típus>-ba tartozó literálokat vagy konstans azonosítókat tartalmaz,vesszővel (,) elválasztva.

A fenti definíció rekurziv, ugyanis a rekordon belüli CASE részek egymásba ágyazhatók. Ezt a gyakorlatot azonban nem javasoljuk, mert teljesen áttekinthetetlen adatstruktúrák jöhetnek létre. Ha a CASE-t a rekord definíciójában használjuk, az **a rekord utolsó mezője lehet** csak. Így bármely rekord-ban egyetlen feltételes mező lehet.

Ha feltételes mezőket nem használunk, akkor a rekord alakja lényegesen leegyszerűsödik:

record <rögzített mezőlista> **end**

ahol a <rögzített mezőlista> egymástól ;-vel elválasztott rögzített mezők definíciójából áll. Egy ilyen definíció alakja:

<mezőnév lista>:<típus>

Ha X rekord típusú változó és név annak a rekordnak egy mezője, akkor arra a mezőre az **X.nev** alakkal tudunk hivatkozni. Ezt az alakot bárhol használhatjuk, ahol az adott mező típusának megfelelő változókat amúgy használhatjuk.

A rekord definíción belül nem használhatjuk a típusfüggvényeket, így a mezők leírásakor használt típusok csak előzőleg már definiált típusok lehetnek. A feltételes rekordban a feltétel leírásaként használt típus csak rendezett (és már definiált) típus neve lehet. A feltételes mezők leírásában szereplő <feltétel lista> elemei csak literálok lehetnek.

Példaként tekintsük az alábbi rekord definíciót

```
type
    konyv =  record
                iro: string[30];
                cím: string[120];
                kiadó: string[45];
                datum: INTEGER;
                ar: REAL
             end
```

s tegyük fel, hogy a programban valahol a **var** miez:konyv; deklarációt használtuk. Ezt követően a miez.iro, miez.cím, miez.kiadó változók sztring változók a típusdeklarációban jelzett hosszal. A miez.datum egész, míg a miez.ar valós típusú változók. A változók ilyen használatára mondjuk, hogy **a rekord elemeire név szerint kell hivatkozni**.

Alkalmanként egy programban fárasztó lehet kiírni folyton azt, hogy miez. Ezért a Pascal tartalmaz egy **with** utasítást, amelyik megmondja, hogy a használt mezőnevek elé melyik rekord típusú változó nevét kell érteni. Ennek az utasításnak az alakja:

with <rekord változó lista> **do** <utasítás>

Természetesen a <rekord változó lista> elemei közt nem lehet két azonos típusú változó. A lista elemeit veszővel kell elválasztani.

A feltételes rekord használatára is adunk egy egyszerű példát:

```
type
    csaladiallapot=(hajadon,notlen,hazas,elvalt,ozvegy)
    adatok=  record
                nev: string[50];
                lakcim: string[120];
                case jelzo: csaladiallapot of
                      hajadon,notlen:();
                      hazas:(hazastars:string[50]);
                      elvalt,
                      ozvegy:(volt hazastarts:string[50];datum: INTEGER)
             end
```

Tegyük fel, hogy az <u>ujbelepo</u> változó <u>adatok</u> típusú. Ebben az esetben az <u>ujbelepo.nev</u>, <u>ujbelepo.lakcim</u> és <u>ujbelepo.jelzo</u> változók értékét a rekord mindig tartalmazza. Az <u>ujbelepo.hazastars</u> csak abban az esetben tartalmaz értékes adatot, ha az <u>ujbelepo.jelzo</u>

nem hajadon vagy notlen. Az ujbelepo.datum pedig csak akkor, ha az ujbelepo.jelzo értéke elvalt vagy ozvegy.

A Pascal lehetőséget biztosít halmazok kezelésére, igaz meglehetősen korlátozott formában. A halmazok elemei ugyanis egy rendezett halmazból kell hogy kikerüljenek, s ezen túlmenően a Turbo Pascal-ban egy halmaznak legfeljebb 256 eleme lehet. A halmaz deklaráció alakja:

set of <típus>;

s mint utaltunk rá a <típus> csak rendezett lehet.

A halmaz típusú változókkal lényegesen több művelet végezhető, mint a tömb vagy rekord típusú változókkal. Ezek közül a leglényegesebb, hogy egy halmazt megadhatunk elemeinek felsorolásával:

[<elem lista>]

Az <elem lista> vagy a <típus>-nak megfelelő kifejezéseket, vagy <első>..<utolsó> alakú kifejezéseket tartalmaz, ahol <első> és <utolsó> természetesen a <típus>-ba tartozó elemek. Ebben az esetben az <első> és <utolsó> közti összes elem (beleértve a határokat is) a halmazba kerül. Egész típusú halmaz megadására mutatunk néhány példát:

```
[1,2,3,4,5]
[1..5]
[1..4,5]
[1..10,20..30]
```

Az első három kifejezés ugyanazt a halmazt adja meg.

Halmazműveletek
Az aritmetikai műveletek jelentése megváltozik, ha halmazokra használjuk:

+	két halmaz uniója;
-	két halmaz különbsége;
*	két halmaz metszete;
=	igaz, ha a két halmaz egyenlő;
<>	igaz, ha a két halmaz nem egyenlő;
<=	igaz, ha az első halmaz része a másodiknak;
>=	igaz, ha az első halmaz tartalmazza a másodikat;
in	igaz, ha az első argumentum eleme a másodiknak.

A Turbo Pascal lehetőséget biztosít dinamikus változók használatára is. A dinamikus változók a program utáni és az adatok előtti memória részben lehet elhelyezni. A dinamikus változók valójában véges gráfok, amelyek csúcsában statikus változók (tömbök, rekordok, halmazok) értékei, illetve a gráf egyéb pontjaira vonatkozó mutatók vannak elhelyezve. A Turbo Pascal utasításai segítségével tetszőleges gráfstruktúra felépíthető, bár ennek komplexitását célszerű bizonyos korlátok alatt tartani. Annál is inkább, mert a Turbo Pascal-nak nincs beépített szemétgyűjtési eljárása, s így a dinamikus memória hamar megtelhet.

A gráfok felépítésére az ún. mutatókat használhatjuk. A mutató egy 2-byte-os szám, ami a memória azon helyére mutat, ahol az általa jelzett érték található. A mutatókkal

önmagában kevés művelet végezhető, helyette az általuk mutatott objektumokon kell a kívánt műveleteket elvégezni. A mutató típus definiálására nem alapszó, hanem a fel nyíl(↑) szolgál:

↑<típus>;

Ha X és Y két azonos típusú mutató, akkor a következő műveleteket végezhetjük velük:

Értékadás a: X:=Y. Az X mutató értéke az Y mutató értékével lesz azonos, azaz mindkettő a memóriának egy és ugyanazon pontjára mutat. Így természetesen az általuk mutatott érték is megegyezik.

Érték kijelölése: A mutató által kijelölt <típus> típusú objektumra az X↑ változóval hivatkozhatunk. A fenti értékadás következtében pl. X↑=Y↑ teljesülni fog.

Értékadás b: X↑:=Y↑. Ennek hatására az Y által mutatott értéknek egy másolatát készíti el a program, s arra fog az X mutató mutatni.

Egyenlőség lekérdezése: Lehetőség van két mutató egyenlőségét ellenőrizni. Mind az X=Y, mind az X<>Y alakot használhatjuk.

Üres mutató: Lehetőség van arra, hogy a mutató ne mutasson semmire. Erre szolgál a nil konstans. Az X:=nil értékadás hatására az X mutató a 'semmibe' mutat. A Turbo Pascal a nil-t úgy valósítja meg, hogy mutató a 0000H címre mutat. A CP/M ismert felépítése miatt a 0000H címen nem lehet értékes adat. Lehetőség van az X=nil, illetve X<>nil relációk használatára is.

A mutató típusú változókat dinamikus változóknak hívjuk, mert fordításkor a rendszer nem foglal nekik helyet a memóriában. Ehhez a programban bizonyos speciális parancsokat kell kiadni. További eljárások biztosítják a már lefoglalt memória felszabadítását. Ezek az eljárások a következők:

New(<mutató>**)**
GetMem(<mutató>,<méret>**)**
Mark(<mutató>**)**
Release(<mutató>**)**
Dispose(<mutató>**)**
FreeMem(<mutató>,<méret>**)**
MemAvail
MaxAvail

A New eljárás egy új mutatót hoz létre, s pontosan annyi byte-ot foglal le a memóriából, amennyire az általa mutatott értéknek szüksége van. A GetMem hasonlóan működik, de a második paraméterben megadott <méret>-nyi byte-ot foglalja le a memóriában. A Dispose felszabadítja a mutató típusú változó által foglalt helyet. Ugyanez a feladata a FreeMem-nek, de hasonlóan a GetMem-hez itt is meg kell adni a byte-ok számát. Ennek azonosnak kell lennie a GetMem-ben szerepelt értékkel. Ellenkező esetben a rendszer nem tudja pontosan kezelni a dinamikus memóriát s az értékek egymásra íródhatnak.

Mint említettük, a Turbo Pascal nem végez szemétgyűjtést, ami nagy mértékben meggyorsítja a mutatókkal végzett műveleteket, de kényelmetlen mellékhatásokkal járhat. Ha pl. a Dispose-zal törlünk egy 10*10-es tömböt, akkor annak megfelelően lyuk

marad a memóriában. Ha gyakran használjuk a mutatókat, akkor a memória olyan lesz, mint a szita, ami végül megint csak lelassítja a rendszert és meg is telhet a memória.

A MemAvail függvény a dinamikus memóriában levő szabad hely nagyságát mutatja byte-okban. A MemAvail az 'összes' lyukat hozzászámolja a szabad területhez. A MaxAvail függvény a leghosszabb összefüggő memóriarész nagyságával tér vissza. Bizonyos esetben ez lényegesebb, mint, hogy összesen mennyi memória van.

A Mark eljárás végrehajtása után az argumentumaként szereplő mutató a dinamikus memória tetejére fog mutatni, vagyis az eddig még egyáltalán nem használt memóriarész elejére. A Release eljárás hatására az argumentumaként szereplő mutató felett levő mutatók és a hozzájuk tartozó objektumok törlődnek. A Mark és a Release együttes használatával egy korlátozott szemétgyűjtés valósítható meg, ami általában elegendő.

A ptr függvény segítségével lehetőség van a mutató által mutatott érték közvetlen megadására. Az addr függvény pedig közvetlenül megadja, hogy a memória melyik részére is mutat.

Az alábbi két program a mutatók legegyszerűbb tulajdonságait illusztrálja:

```
program pointer_pelda1;
var
      p:↑INTEGER;
      n:INTEGER;

begin
      n:=111;                  {n kezdőértékének beállítása}
      new(p);                  {a p mutató létrehozása}
      p:=nil;
      writeln(addr(p↑));       {=0, mert nil a 0000h-ra mutat}
      p:=ptr(addr(n));         {p most az n-re mutat}
      writeln(p↑);             {=111, a p által mutatott érték kiírása}
      repeat                   {várakozás egy billentyű}
      until keypressed;        { megnyomására}
end.
```

```
program pointer_pelda2;
var
      p:↑INTEGER;
      n:INTEGER absolute $a000; {n deklarálása a $a000 címre}

begin
      new(p);                  {a p mutató létrehozása}
      p:=ptr($a000);           {a p a $a000 címre, azaz n-re mutat}
      n:=123;                  {n értékének beállítása}
      writeln(p↑);             {=123, a p által mutatott érték kiíratása}
      repeat                   {várakozás egy billentyű}
      until keypressed;        {megnyomására}
end.
```

A Turbo Pascal a Standard Pascal-tól gyökeresen eltérő módon kezeli a file típust. A Turbo-ban a file típus megadása egyszerű:

file of <típus>;

A <típust> a file típus alaptípusának hívjuk. A file típusú változókkal pontosan azok a műveletek végezhetők el, amik alaptípusú változókkal, két lényeges eltéréssel:

- a file változókhoz konkrét lemezes file-ok rendelhetők;
- lehetőség van a file változó tartalmát a hozzá rendelt lemezes file meghatározott helyére kiírni, illetve onnan visszaolvasni.

13.6 Programstruktúrák

Ebben a részben ismertetjük a a Turbo Pascal programokban használható (struktúrált) utasításokat. A <végrehajtható rész> egyetlen ilyen utasításból áll.

Üres utasítás A Turbo Pascal az utasítások elválasztására a pontosvesszőt (;) használja. Két egymás utáni, vagy egy felesleges pontosvesszőt ezért a fordító üres utasításnak tekint. Célszerű azonban ennek használatát elkerülni, mert alkalmanként rosszul struktúrált programokat eredményezhet.

Címkés utasítás A Turbo Pascal lehetőséget nyújt címkék használatára is. A címkét az utasítástól kettősponttal (:) kell elválasztani. Az utasítás alakja tehát:

<címke>:<utasítás>

Értékadó utasítás Az értékadó utasítás alakja:

<struktúrált változó>:=<kifejezés>

A <struktúrált változó> vagy egyszerű változó, vagy tömbváltozó vagy rekord komponens vagy mutató. A kifejezés típusának meg kell egyeznie a struktúrált változó típusával. A kifejezés a Turbo Pascal beépített műveleteiből, vagy már definiált (deklarált) függvényekből épülhet fel.

Eljáráshívás Az eljáráshívás alakja:

<eljárásnév>(<aktuális paraméter lista>)

Az <eljárás név> a Turbo Pascal előre definiált eljárásainak egyike lehet, vagy a programban már szereplő eljárás kell hogy legyen. Az aktuális paraméterek számban és típusban meg kell hogy egyezzenek az eljárás deklarációjában szereplő formális paraméterekkel. Ha valamelyik formális paraméter név szerinti hivatkozás, akkor a neki megfelelő aktuális paraméter csak egyszerű változó lehet.

Összetett utasítás Lehetőség van több utasítás összevonására utasítászárójelek használatával. Az utasítás alakja:

begin <utasítássorozat> **end**

Az <utasítássorozat> egymástól pontosvesszővel elválasztott utasításokat tartalmaz. Az utasítások a leírás sorrendjében hajtódnak végre.

Ugró utasítás A Pascal egyetlen nem struktúrált utasítást tartalmaz. Ennek alakja GOTO <címke> alakú, ahol a <címke> a programban már deklarált címke kell hogy legyen. A program a <címke> címkével ellátott utasítással fog folytatódni. Struktúrált utasítás belsejébe be-, vagy onnan kilépni a GOTO utasítással tilos!

Feltételes utasítások A feltételes utasítás alakja:

IF <kifejezés> **THEN** <utasítás> {**ELSE** <utasítás>}

Az utasítás alakjából is látszik, hogy abban az esetben, ha a THEN vagy ELSE rész több utasításból áll, akkor azt utasítászárójelekkel (begin,end) kell zárni!

A Turbo egy másik feltételes utasítást is tartalmaz, amelyik lehetővé tesz többirányú programelágazást is. Ennek alakja a következő:

CASE <kifejezés> **OF** <eset-utasítás lista> {**ELSE** <utasítássorozat>} **END**

A <eset-utasítás lista> egymástól pontosvesszővel (;) elválasztott eset-utasításokat tartalmaz. Egy eset-utasítás alakja:

<eset-lista>:<utasítás>;

ahol az <eset-lista> egymástól vesszővel (,) elválasztott literálokat tartalmaz. A program kiértékeli a kifejezés értékét, majd megkeresi az első literált, amelyik ezzel egyenlő, s az annak megfelelő utasítást hajtja végre. Ha nem talál ilyent, akkor az ELSE ág kerül végrehajtásra (ha van). Abban az esetben, amikor az eset-utasítás maga is több utasításból áll, akkor utasítás zárójelek közé kell tenni. Ez az ELSE részre nem vonatkozik.

Ciklus utasítások A Turbo Pascal háromféle ciklust ismer. Ezek sorra a következők:

A/

FOR <változó>:= <kezdőérték> { **TO** | **DOWNTO** } <végérték> **DO** <utasítás>

B/

REPEAT <utasítássorozat> **UNTIL** <kifejezés>

C/

WHILE <kifejezés> **DO** <utasítás>

Az A/ struktúrált utasítás gyakorlatilag azonos a BASIC FOR ciklusával, míg a B/ és C/ a C-128 BASIC DO...LOOP ciklusának felel meg.

A FOR ciklusban a <változó> tetszőleges **rendezett** típusú változó lehet. Szemben tehát a legtöbb Commodore BASIC-kel, nem lehet valós! A <kezdőérték> és <végérték> ugyanilyen típusú kifejezések, amelyek értéke egyszer kerül kiszámításra. A ciklusváltozó felveszi a kezdőértéket, majd ezzel lefut a DO után álló utasítás (a ciklusmag). A ciklusváltozó értéke - a rendezett típusnak megfelelően - eggyel megnő, vagy csökken. Ez attól függ, hogy a TO (felfelé) vagy a DOWNTO (lefelé) alapszót használtuk. A ciklus még a végértékkel is lefut. A BASIC-kel ellentétben, ha a kezdőérték már eleve meghaladta a végértéket, **a ciklus egyetlen egyszer sem fut le**.

A WHILE ciklus mindaddig fut, amíg a <kifejezés> igaz. Ha a <kifejezés>-t a rendszer hamisnak találta, akkor az <utasítás> már nem kerül végrehajtásra. A WHILE-ban nincs ciklusváltozó. Ha ilyenre mégis szükség van, akkor azt a WHILE utasítás előtt kell beállítani, s a ciklusmagban módosítani.

A REPEAT ciklusban levő <utasítássorozat> utasításait pontosvesszővel (;) kell elválasztani egymástól. A ciklus addig ismétlődik, míg a <kifejezés> hamis. Amikor először igazzá válik, a ciklus lejár. Szemben a WHILE-lal, a REPEAT ciklusmagja legalább egyszer lefut. A REPEAT - hasonlóan a WHILE-hoz - nem tartalmaz ciklusváltozót, azt magunknak kell beállítani, illetve módosítani.

Gépi kódú programrész beillesztése A Turbo Pascal fordítóprogram gyorsaságát - túl azon, hogy zseniális programozók készítették - két dolognak köszönheti. Az egyikről már szóltunk: típusos nyelv, s ráadásul nem tartalmaz szemétgyűjtési eljárást. A másik,

hogy nem a hagyományos fordítók elvén működik, a fordítást nem követi egy szerkesztési fázis. Ennek egyik következménye, hogy nem lehet gépi kódú rutinokat a programhoz hozzászerkeszteni. Ezen a rendszer úgy segít, hogy viszont a program szövegébe gépi kódú rutinok építhetők be. Erre szolgál az **inline** utasítás, amelyiket egy lista követ. A lista minden eleme a gépi kódú rutin egy byte-ja. Ezt a byte-ot a következő kifejezésekkel adhatjuk meg:

Kifejezés	Jelentés
<előjel nélküli egész>	közvetlenül megadja a byte-ot
<konstans azonosító>	közvetlenül megadja a byte-ot
<változó>	a program a változó címét a rutin hívása előtt ide másolja
*	a processzor utasításszámlálója (ahová a gépi kódu program kerül)
*<előjel><literál>	a processzor utasításszámlálójának megfelelően módosított értéke (az <előjel> + vagy - lehet)

(Emlékeztetünk rá, hogy a <literál> vagy esetleg előjeles <konstans azonosító> vagy előjeles szám vagy konstans sztring.)

Az inline utasítás alakja:

INLINE (<byte sorozat>**);**

ahol a <byte sorozat> elemei a fenti táblázatban szereplő elemek lehetnek a törtjellel (/) elválasztva. A fordító ebből egy gépi kódú rutint készít. Ennek meghívása előtt a változóként megadott elemek értékét behelyettesíti, majd meghívja a rutint. Visszatéréskor nincs paraméter átadás. Ha erre szükség van, akkor valamelyik statikus változót célszerű erre felhasználni. Ennek kezdőcime az addr előre definiált függvénnyel átadható a gépi kódú alprogramnak.

Hosszabb betéteket célszerű valamelyik makro assemblerrel megírni, majd a már letesztelt programot hexadecimális alakban kilistázni, s utána beírni a programba.

13.7 Előre definiált eljárások és függvények

Ebben a részben felsoroljuk a Pascal 3.1-es verziójában használható előre definiált változókat, eljárásokat és függvényeket funkció szerinti csoportosításban. Közös jellemzőjük, hogy átdefiniálhatók, de azt követően az eredetileg definiáltak már nem használhatók.

Előre definiált változók

Szöveg file-ok

Aux	RS232-es csatorna
Con	CP/M konzol egység
Input	standard input periféria
Kbd	CP/M konzol input egység
Lst	CP/M listázó periféria
Output	standard output periféria
Trm	CP/M konzol egység

A Con és Kbd input esetén eltérő hatású. A Con használatakor a program egy teljes sort olvas be, amit előzetesen még meg is szerkeszthetünk. Ha az input pufferben még voltak karakterek, az elvész. A Con használatakor a beírt karakterek automatikusan visszaíródnak a képernyőre. A Kbd egyszerre egyetlen karaktert olvas, s az nem íródik automatikusan vissza a képernyőre.

Az AUX: CP/M eszköz a C128-as CP/M-ben nincs installálva, használata ekvivalens a NULL: eszköz használatával.

Mem[]
array[$0000..$FFFF] of byte; a processzor memória területe

BufLen
integer; a READ utasítás pufferének a hossza

HeapPtr
mutató; a dinamikusan allokálható memória tetejére mutat

RecurPtr
mutató; a rekurzió verem tetejére

StackPtr
mutató; a verem mutatója

Képernyő és billentyűzet kezelés

ClrEol
eljárás: törlés a kurzortól a sor végéig

ClrScr
eljárás: a képernyő teljes törlése

CrtExit
eljárás: a képernyőre a termináló sztringsorozat elküldése

CrtInit
eljárás: a képernyőre az inicializáló jelsorozat elküldése

DelLine
eljárás: a kurzor sorának törlése

GoToXY(x,y:integer)
eljárás: a kurzor az x. sor y. oszlopába kerül

HighVideo
eljárás: a következő kijelzések nagy intenzitással jelennek meg

InsLine
eljárás: beszúrás a kurzor helyén

IOresult
függvény: az IO hiba kódjával tér vissza

KeyPressed
függvény: TRUE-val tér vissza, ha van karakter az input pufferban

LowVideo
eljárás: a következő kiírások alacsony intenzitással történnek

NormVideo eljárás: hatása azonos a HighVideo-val

File-kezelő utasítások

Assign(f:file;nev:string)
Az f file típusú változóhoz a nev karaktersorozatot rendeli. A 'nev' nevű file-ra ezek után az f változóval lehet hivatkozni.

BlockRead(f:file;var;db:integer)
BlockRead(f:file;var;db,kod:integer)
eljárás: az f - előzőleg már megnyitott file-ból beolvassa a következő db darab 128byte-os blokkot és a memóriába helyezi Addr(var)-tól kezdve. A var változó típusa érdektelen. Ha a kod egész változót is megadjuk, akkor a kod értékébe visszatéréskor a ténylegesen beolvasott rekordok száma kerül.

BlockWrite(f:file;var;db:integer)
BlockWrite(f:file;var;db,kod:integer)
eljárás: az f - előzőleg már megnyitott file-ba kiírja az Addr(var)-tól kezdődő pufferból a következő db darab 128byte-os blokkot. A var változó típusa érdektelen. Ha a kod egész változót is megadjuk, akkor a kod értékébe visszatéréskor a ténylegesen kiírt rekordok száma kerül

Chain(f:file)
eljárás: betölti és lefuttatja a .CHN kiterjedésű, az f file típusú változóval azonosított, a Turbo Pascal CHN opciójával fordított programot

Close(f:file)
eljárás: lezárja az f file típusú változóhoz tartozó file-t.

Eof(f:file)
függvény: TRUE-val tér vissza, ha a file végét elértük.

EoLn(f:text)
függvény: TRUE-val tér vissza, ha a szöveges file sorát már végigolvastuk.

Erase(f:file)
eljárás: törli az f file változóval megadott file-t a lemezről.

Execute(f:file)
eljárás: betölti és lefuttatja az f file változóval azonosított COM kiterjesztésű file-t.

FilePos(f:file)
egész függvény: a file író/olvasó fejének pozíciójával tér vissza. A következő író/olvasó utasítás ennél a pozíciónál kezdődik. Szöveges file-re nem ad jó értéket.

Read([f:file,]var1[,var2...])
eljárás: hatására az f file-ról vagy eszközről (hiányában az Input nevű standard beviteli perifériáról) olvas be adatokat a program, s azokat sorban a var1, var2 stb. változókhoz rendeli.

ReadLn([f:text,]var1[,var2...])
eljárás: hatására az f file-ról (hiányában az Input nevű standard beviteli perifériáról) olvas be adatokat a program, s azokat sorban a var1, var2 stb. változókhoz rendeli.

Rename(f:file,ujnev:string)
eljárás: hatására az f file új neve az 'ujnev' sztringváltozó pillanatnyi tartalma lesz.

Reset(f:file)
eljárás: létező file megnyitása.

Rewrite(f:file)
eljárás: új file létrehozása és megnyitása. Ha az adott file már létezik, akkor a parancs 0 hosszúságúra csonkolja.

Seek(f:file;pos:integer)
eljárás: az adott file író/olvasó fejét az n-ik rekordra állítja. A következő író/olvasó művelet innen kezdődik. Szöveges file-ra nem jól működik.

Write([f:file,]var1[,var2...])
eljárás: hatására az f file-ba vagy eszközre (hiányában az Output nevű standard kiviteli perifériára) írja ki a var1, var2 stb. változókhoz rendelt adatokat.

Write([f:text,]var1[,var2...])
eljárás: hatására az f file-ba vagy eszközre (hiányában az Output nevű standard kiviteli perifériára) írja ki a var1, var2 stb. változókhoz rendelt adatokat. Ezután még elküld egy kocsivissza/soremelés karaktersorozatot is. Csak szöveges file-ra használható.

Egyéb utasítások

Addr(var)
egész függvény: a var tetszőleges típusú változó címével tér vissza.

FillChar(mem,db:integer;ch:char)
eljárás: a mem memóriacímtől kezdve db darab byte-ot tölt fel a ch karakterek. A ch típusa byte is lehet.

Hi(v:integer)
byte függvény: a v egész típusú változó felső byte-jával tér vissza.

Lo(v:integer)
byte függvény: a v egész típusú változó alsó byte-jával tér vissza.

Move(mem1,mem2,db:integer)
eljárás: a mem1 címtől kezdődően db darab byte-ot másol át a mem2 címtől elhelyezve.

SizeOf(var)
egész függvény: tetszőleges típusú változó által foglalt byte-ok számával tér vissza.

Swap(v:integer)
egész függvény: megcseréli a v változó alsó és felső byte-ját, s az úgy kapott értékkel tér vissza. Pl. Swap($ABEF)=$EFAB.

Az operációs rendszer funkcióinak használata

Bdos(fk:integer[;Creg:integer])
eljárás: az fk-ik Bdos funkció meghívása. A rutin meghívása előtt az opcionális második paramétert a C regiszterbe tölti a program.

Bdos(fk:integer[;Creg:integer])
függvény: az fk-ik Bdos funkció meghívása. A rutin meghívása előtt az opcionális második paramétert a C regiszterbe tölti a program. A BDOS funkció után az A regiszter tartalmával tér vissza.

BdosHL(fk:integer[;Creg:integer])
függvény: az fk-ik Bdos funkció meghívása. A rutin meghívása előtt az opcionális második paramétert a C regiszterbe tölti a program. A BDOS funkció után a HL regiszter tartalmával tér vissza.

Egyebek

Delay(t:integer)
eljárás: a program futását t ezredmásodpercre felfüggeszti.

Halt
eljárás: a program megáll.

absolute

deklaráció: a változó tárolására szolgáló cím, konstansként megadható. Használata: <név>:<típus> absolute <cím>.

forward

deklaráció: az eljárás törzsét helyettesíti. Különösen akkor használható, ha az .INC file-ban olyan eljárást írunk, amelyik az .INC-et tartalmazó file-ban lesz majd deklarálva.

13.8 Fordítási opciók

A Turbo Pascal programokba lehetőség van úgynevezett fordítási opciók beillesztésére, amelyek a fordítás menetét vezérlik, s nem részei az 'igazi' programnak. A Turbo-ban a fordítási opciókat kapcsos zárójelek közé kell tenni s mindig egy $ jellel bevezetni: **{$...}**. A kapcsos zárójelek a megjegyzéseket tartalmazzák. Ha a megjegyzés egy $ jellel kezdődik, a fordító azt fordítási opciónak értelmezi, s végrehajtja. Megjegyzéseket a (*, illetve *) jelek közé is lehet tenni. Ebben az esetben azonban a fordító nem hajtja végre az esetleg itt elhelyezett vezérlő karaktereket! A kapcsos zárójelek a <CTRL-[>, illetve <CTRL-]> billentyűzéssel állíthatók elő. A 40 oszlopos képernyőn két grafikus jel látható, a 80 oszlopos képernyőn azonban igazi kapcsos zárójelek.

Egy kapcsosszárójel párban több fordítási opciót is megadhatunk. A $ jelet csak egyszer kell szerepeltetni, s az opciókat vesszővel kell elválasztani egymástól: {$I-,V-}. A $ jel után nem lehet szóköz. Az egyes opciók a következők lehetnek:

B± I/O mód kiválasztása Kezdeti érték: B+
A B fordítási opció a használt konzolt adja meg. A B+ opció esetén a standard INPUT és OUTPUT utasítások a CON: logikai eszközre vonatkoznak. Ellenkező (B-) esetben pedig a TRM: logikai eszközre.

C± <CTRL-S> és <CTRL-C> Kezdeti érték: C+
A C opció engedélyezi/letiltja a ki/beviteli utasítások megszakítását a <CTRL-C>, illetve a <CTRL-S> (=<NO SCROLL>) billentyűkkel. Ha engedélyezzük, akkor az I/O műveleteket a <CTRL-C> megszakítja, s a <CTRL-S> megállítja/újraindítja a képernyőre történő írást. A C+ opció engedélyezi ezen billentyűk használatát.

I± I/O hibakezelés Kezdeti érték: I+
Saját I/O hibakezelést az I- opcióval engedélyezhetünk. Ekkor I/O hiba esetén a program fut tovább, s az IOResult függvény segítségével lekérdezhetjük, milyen hiba történt. Az I+ opció használata esetén I/O hiba esetén a program futása félbeszakad.

I <filenév> szöveg beillesztése
Az opció lehetőséget biztosít a programok könyvtárszerű kezelésére. A <filenév> nevű szöveges file-t a fordító úgy tekinti, mintha a program része lenne, s teljes egészében belefordítja a programba. A fordítási opció nem ágyazható önmagába, azaz a <filenév> nevű program nem tartalmazhat további I <filenév1> alakú opciót.
A <filenév> után egy szóköznek kell állnia, különben a fordító nem tudja a file nevét helyesen értelmezni:

```
{Itipus.inc }
{Irutin12.inc ,C+}
```

A könyvtári programoknak általában az INC típust szokás adni, ezzel is jelezve, hogy csak egy nagyobb program részeiként értelmezhetők. Ha az opcióban nem adjuk meg a file-név kiterjesztését, akkor azt a fordító automatikusan PAS-nak tekinti.

R± index ellenőrzés Kezdeti érték: R-
A R opcióval a programba fordíthatjuk azt a rutint, amelyik ellenőrzi, hogy a tömbök indexei a megfelelő határok közé esnek-e. Ez a rutin ellenőrzi azt is, hogy a résztípusok értékei a megengedett határok között mozognak-e. Elsősorban a program fejlesztéskor célszerű használni, lényegesen lassítja a program futását.

V± sztringhossz ellenőrzés Kezdeti érték: V+
Az opció segítségével be/ki kapcsolhatjuk a sztringhossz ellenőrzését. Erre akkor kerül sor, ha egy eljárásnak vagy függvénynek név szerint (var-ral) adunk át egy sztring paramétert. A V+ opció használatakor csak akkor nem kapunk hibajelet, ha a formális és aktuális paraméter sztring lehetséges maximális hossza megegyezik. (Ez az a hossz, amit a két paraméter deklarálásakor megadtunk.)

U± <CTRL-C> engedélyezése Kezdeti érték: U-
Az opció engedélyezi/letiltja a <CTRL-C> tetszőleges pillanatban történő használatát. A C opcióval szemben ez az opció bármikor engedélyezheti a program futásának megszakítását. Elsősorban a tesztelési munka közben célszerű engedélyezni. Ez ugyanis az egyetlen lehetőség, hogy egy végtelen ciklusból – a rendszer újratöltése nélkül – ki lehessen lépni.

A± abszolut kód Kezdeti érték: A+
Az opció lehetővé teszi rekurziv eljárások tiltását és engedélyezését. Az A+ opció abszolút kódot generál, az így fordított eljárások, függvények rekurzivan nem használhatók. A rekurzív eljárások előtt az A- opciót kell használni. Az eljárás végén újra visszatérhetünk abszolút kódra az A+ opcióval. Ez csökkenti a program méretét.

Wn WITH mélysége Kezdeti érték: W2
Az opció az egymásba ágyazott DO utasítások mélységét adja meg. n előjel nélküli szám, maximum 9 lehet. Ha a W opcióban megadottnál több with utasítást skatulyázunk egymásba, akkor hibaüzenetet kapunk.

X± Tömb optimalizálás Kezdeti érték: X+
A fordító optimalizálja a tömbök elhelyezkedését és elérésüket. Ez meghosszabbítja a fordítási időt, s valamivel a lefordított programot is. Az opció az X- használatával kikapcsolható.

13.9 Turbo Pascal hibaüzenetek

<u>Fordítási hibák</u>

Az alábbiakban felsoroljuk a fordítás közben kapható hibajelzéseket. Ha a Turbo Pascal indításakor a hibaüzenetek szövegét is betöltöttük, akkor a számok után a hiba szövege is kiíródik. Az <ESC> billentyű megnyomása után visszatérhetünk a programszerkesztőbe s a kurzor a hiba helyén jelenik meg.

01 ';' expected
A fordító ';' helyett más jelet talált.

02 ':' expected
A fordító ':' helyett más jelet talált.

03 ',' expected
A fordító ',' helyett más jelet talált.

04 '(' expected
A fordító '(' helyett más jelet talált.

05 ')' expected
A fordító ')' helyett más jelet talált.

06 '=' expected
A fordító '=' helyett más jelet talált.

07 ':=' expected
A fordító ':=' helyett más jelet talált.

08 '[' expected
A fordító ']' helyett más jelet talált.

09 ']' expected
A fordító ']' helyett más jelet talált.

10 '.' expected
A fordító '.' helyett más jelet talált.

11 '..' expected
A fordító '..' helyett más jelet talált.

12 BEGIN expected
A fordító BEGIN helyett más jelet talált.

13 DO expected
A fordító DO helyett más jelet talált.

14 END expected
A fordító END helyett más jelet talált.

15 OF expected
A fordító OF helyett más jelet talált.

17 THEN expected
A fordító THEN helyett más jelet talált.

18 TO or DOWNTO expected
A fordító TO vagy DOWNTO helyett más jelet talált.

20 Boolean expression expected
A fordító logikai kifejezés helyett valami mást talált.

21 File variable expected
Az adott helyen csak file típusú változót lehet használni.

22 Integer constant expected
Az adott helyen csak egész típusú konstanst lehet használni.

23 Integer expression expected
Az adott helyen csak egész kifejezés használható.

24 Integer variable expected
Az adott helyen csak egész változó használható.

25 Integer or real constant expected
Egész vagy valós típusú konstans helyett valami mást talált a rendszer.

26 Integer or real expression expected
Egész vagy valós típusú kifejezés helyett valami mást talált a rendszer.

27 Integer or real variable expected
Egész vagy valós típusú változó helyett valami mást talált a rendszer.

28 Pointer variable expected
Ezen a helyen csak mutató használható.

29 Record variable expected
Ezen a helyen csak record típusú változó használható.

30 Simple type expected
Csak rendezett típus vagy típus azonosító használható ezen a helyen.

31 Simple expression expected
A kifejezés nem tartalmazhat zárójelet.

32 String constant expected
Ezen a helyen egy sztring literálnak kell állnia.

33 String expression expected
A fordító ezen a helyen sztring kifejezést várt, de valami mást talált.

34 String variable expected
Sztring változó helyett valami mást talált a rendszer.

35 Textfile expected
Ezen a helyen csak textfile használható.

36 Type Identifier expected
Ezen a helyen csak típus azonosító használható.

37 Untyped file expected
Ezen a helyen csak típus nélküli file használható.

40 Undefined label
Olyan címkét használtunk, ami nem volt label utasítással definiálva.

41 Unknown identifier or syntax error
Ismeretlen azonosítót használtunk, de lehet, hogy csak rosszul írtuk le a programot.

42 Undefined pointer type in preceding type definitions
A mutató alaptípusát nem definiáltuk sehol.

43 Duplicate identifier or label
Kétszer definiáltunk eg azonosítót vagy címkét.

44 Type mismatch
Az adott helyen nem megengedett típust használtunk.

45 Constant out of range
A konstans kívül esik az értékhatárokon.

46 Constant and CASE selector type does not match
A konstans és a CASE-ben megadott típus nem egyezik.

47 Operator type(s) does not match operator
Az argumentumok típusa nem felel meg a használt műveletnek.

48 Invalid result type
Az eredmény nem megfelelő típusú.

49 Invalid string length
Nem megengedett hosszúságú sztring.

50 String constant length does not match type
A használt sztring literál túl hosszú.

51 Invalid subrange base type
A résztípus képzést csak rendezett típusra lehet használni, másra kíséreltük meg.

52 Lower bound > upper bound
A résztípus definíciójában a kezdeti érték nagyobb mint a végérték.

53 Reserved word
Alapszót akartunk újra definiálni.

54 Illegal assignment
Nem megengedett értékadás, pl. konstans változóra.

55 String constant exceeds line
A Turbo Pascal egy esetben érzékeny a sor végére: a sztring konstans nem nyúlhat túl rajta. A fordító ilyent talált.

56 Error in integer constant
Az egész konstans kiértékelése közben hiba történt, pl. nem megengedett karaktert használtunk.

57 Error in real constant
A valós konstans kiértékelése közben hiba történt, pl. nem megengedett karaktert használtunk.

58 Illegal character in identifier
Azonosítóban vagy címkében nem megengedett karaktert használtunk.

60 Constants are not allowed here
Ezen a helyen nem lehet konstanst használni.

61 Files and pointers are not allowed here
A programban ezen a helyen file-t vagy mutatót használtunk, holott nem lehet.

62 Structured variables are not allowed here
Struktúrált változót nem használhatunk ezen a helyen.

63 Textfiles are not allowed here
Ezen a helyen nem használhatunk textfile-t.

64 Textfiles and untyped files are not allowed here
Ezen a helyen nem használhatunk textfile-t és típus nélküli file-t.

65 Untyped files are not allowed here
Ezen a helyen nem használhatunk típus nélküli file-t.

66 I/O not allowed here
I/O művelet ezen a helyen (vagy módon) nem végezhető.

67 Files must be VAR parameters
Eljárások és függvények formális paraméterlistáiban a file típusú változókat csak név szerint (var) használhatjuk.

68 File components may not be files
A file alaptípusa nem tartalmazhat file-t.

69 Invalid ordering of fields
A mezők sorrendje nem megfelelő.

70 Set base type out of range
A set típus alaphalmaza kívül esik a lehetséges tartomány nagyságon (256).

71 Invalid GOTO
A GOTO utasítás ebben a formában nem használható. (GOTO-val nem léphetünk bele egy struktúrába.)

72 Label not within current block
A címke nincs a feldolgozás alatt álló program blokkban.

73 Unidentified FORWARD procedure(s)
A FORWARD-dal definiált eljárás(ok) definíciója hiányzik.

74 INLINE error
Az INLINE utasítás elemei közt hibás van.

75 Illegal use of ABSOLUTE
Az ABSOLUTE utasítást meg nem engedett helyen használtuk. (Pl. név szerinti hivatkozásnál.)

90 File not found
Az I fordítási opcióban megadott file nincs a lemezen.

91 Unexpected end of source
A tárgyprogram véget ért, holott a fordító még nem találta meg a programstruktúra végét.

97 Too many nested WITHs
A Wn opcióban megadott értéknél több WITH utasítást ágyaztunk egymásba.

98 Memory overflow
A program nem fér el a memóriában.

99 Compiler overflow
A fordító munkaterülete megtelt. Csökkenteni kell az egymásba ágyazott programstruktúrák számát s egyszerűsíteni a programot.

Futási hibák

01 Floating point overflow
Valós típusú műveletek közben túlcsordulás történt.

02 Division by zero attempted
0-val kíséreltünk meg osztani.

03 Sqrt argument error
Négyzetgyököt negatív számból nem lehet vonni.

04 Ln argument error
A logaritmusfüggvény argumentuma csak pozitív szám lehet.

10 String length error
Valamelyik sztring művelet eredménye túl nagy sztringet eredményezett.

11 Invalid string index
A sztring indexe túlmutat a sztring hosszán.

90 Index out of range
Az index, amit használtunk kívül van a megengedett értékeken.

91 Scalar or subrange out of range
A rendezett- vagy résztípus típusú változó értéke a megengedett értékhatárokon kívül esik.

92 Out of integer range
Az egész kifejezés értéke már nem esik a gépben ábrázolható értékhatárok közé.

F0 Overlay file not found
Az átlapoláshoz szükséges file-t a rendszer nem tudja betölteni.

I/O hibák

01 File does not exist
A használni kívánt file nem létezik.

02 File not open for input
A file input műveletekre nem használható.

03 File not open for output
A file output műveletekre nem használható.

04 File not open
A használni kívánt file-t még nem nyitottuk meg.

10 Error in numeric format
A write eljárásban használt formátum leírás nem megfelelő.

20 Operation not allowed on a logical device
A végrehajtandó művelet logikai eszközre nem alkalmazható.

21 Not allowed in direct mode
Direkt módon nem alkalmazható.

22 Assign to std files not allowed
Az assign eljárás a standard INPUT és OUTPUT file-ra nem alkalmazható.

90 Record length mismatch
Egy létező file rekord hossza nem azonos a programban megadottal.

99 Unexpected end-of-file
Az utolsó rekord beolvasása közben elérte a rendszer a file végét. Valószínüleg (logikailag vagy fizikailag) sérült a file.

F0 Disk write error
Lemez írási hiba. Valószínüleg a lemez vagy lemezegység fizikai hibája.

F1 Directory is full
A lemez katalógusa megtelt.

F2 File size overflow
Túl sok rekord.

F3 Too many open files
Az adott számú lemezes file nem nyitható meg.

FF File disappeared
Egy előzőleg már sikerrel megnyitott file nincs a lemezen. Lehet, hogy a lemezt közben kicseréltük.

14. fejezet

Turbo Pascal példaprogramok

Ebben a fejezetben Turbo Pascal példaprogramokat adunk. Egyes példák olyanok, hogy bármilyen gépen futó Turbo Pascal-ban jól működnek, mások csak CP/M-es Turbo Pascalban, megint mások csak a Commodore 128-on.

FACTOR
A program az n szám faktoriálisát, n!-t számítja ki. n! = n * (n-1) * ... * 3 * 2 * 1. Az n! érték kiszámítására egy while ciklusban kerül sor. Ha n=0 értéket adunk be, a program megáll.

RFACTOR
Az előbbi feladatot oldja meg, azonban rekurzív eljáráshívással. A használt képlet: n>1 esetén n! = n * (n-1)!.

QUEENS
A program 8 királynőt állít fel a sakktáblán, anélkül, hogy ütnék egymást. A végeredmény az A, B stb. oszlopokban való pozícióját jelzi a királynőknek. Az 15863724 sorozat ezért az A1, B5, C8, D6, E3, F7, G2, M4 felállásnak felel meg.

HANOI
A program a Hanoi tornyai néven ismert játék megoldását mutatja be. Induláskor az első pálcán levő korongok számát kell megadni. A megoldáshoz ez a program is rekurzív eljárásokat használ.

STACK
A program azt mutatja be, hogy hogyan lehet mutatók segítségével vermet szervezni. A deklarált eljárások a verem kreálását (CREATE), a verembe való tételt (PUSH), a veremből való kivételt (POP), s a verem üres voltának ellenőrzését (IS EMPTY) szolgálják. A főprogram ezek egyszerű használatát adja: az általunk megadott számig a verembe rakja az egész számokat, majd visszaolvassa azokat.

CPMDIR
A program a lemez katalógusában szereplő neveket írja ki a képernyőre. A program jól mutatja, hogyan kell Turbo Pascalból a BDOS funkciókat felhasználni.

GRAPHIC
A program a 80 oszlopos képernyő grafikai lehetőségeit mutatja be. A program két INC file-t tartalmaz. Ezek szerepe, hogy a rendszerhez tartozó gépi kódú rutinokat a $E000 címtől betöltse.

Ehhez a szükséges gépi kódú programot – amit a 10. fejezetben adtunk meg, a MAC makro assemblerrel le kell fordítani. Az így kapott GRAPHIC.HEX file-t az INSTALL.INC program tudja a memóriába tölteni. A GRAPHIC.INC tartalmazza a gépi kódú rutinok Turbo Pascalbeli EXTERNAL definícióit is, s ez a programrész hívja meg a INSTALL CODE eljárást is.

Maga a GRAPHIC.PAS program tartalmazza még a ClrScr80 eljárás definícióját, ami törli a 80 oszlopos grafikus képernyőt.

A főprogram egy egyszerű ábrát rajzol a 80 oszlopos képernyőre. (Ha két monitort használunk a rendszerben, vagy egyet, de az átkapcsolható, akkor a 40 oszlopost használjuk a rendszer és a Turbo számára, a 80 oszloposon csak rajzoljunk!)

```
program factor;

{ ============================== }

var n,m,s: integer;
label     ismetles;

{ ============================== }

begin {factor}

  readln(n);
  ismetles: m:=n; s:=n;
  while n>1 do
     begin
       n:=n-1;
       m:=m*n
     end {do};
  writeln(s:1,'!=',m:4);
  readln(n);
  if n<>0 then goto ismetles

end. {factor}
```

```
program rfactor;

var n : real;

{ ============================================ }

function ihaj(ni : real) : real;

begin { ihaj }
  if (ni <= 1) then ihaj := 1
             else ihaj := ni * ihaj(ni - 1)
end;  { ihaj }

{ ============================================ }

begin { rfactor }
  repeat
    readln(n);
    writeln(n:1:0,'! = ',ihaj(n):1:0);
    writeln;
  until (n = 0);
end.  { rfactor }

{ ============================================ }


program queens;

var
   n, k, db : integer;
   x     : array[ 0.. 7] of integer;
   col   : array[ 0.. 7] of Boolean;
   up    : array[-7.. 7] of Boolean;
   down : array[ 0..14] of Boolean;
   ch : char;
{ ====================================== }

{$a-}
procedure generate;
var
   h : integer;

begin
   h := 0;
   repeat
     if (col[h] and up[n-h] and down[n+h])
       then begin
              x[n] := h;
              col[h]     := false;
              up[n-h]    := false;
              down[n+h] := false;
              n := succ(n);
              if (n = 8) then begin
```

```
                                write(db:3,'   ');
                                k := 0;
                                repeat
                                  write(x[k]+1:1);
                                  k := succ(k)
                                until (k = 8);
                                writeln;
                                db := succ(db)
                              end
                         else generate;
              n := pred(n);
              down[n+h] := true;
              up[n-h]   := true;
              col[h]    := true
            end;
      h := succ(h)
   until (h = 8)
end;
{$a+}

{ =================================== }

begin
   n  := 0;
   db := 1;
   clrscr;

   k := 0;
   repeat
      col[k] := true;
      k      := succ(k)
   until k=8;

   k := 0;
   repeat
      up[k-7] := true;
      down[k] := true;
      k       := succ(k)
   until k=15;

   generate;

   writeln;
   writeln('Nyomjon meg egy billenntyut!');
   repeat
   until keypressed;
   read(Kbd,ch)

end.

{ =================================== }
```

```
program hanoi;

var n     : byte;
   lepes : integer;
   ch    : char;
{ ============================================================ }

{$a-}                              { csak CP/M gepen: rekurziv kod }

{ ***** HANOI definicioja ***** }

procedure hanoi_rec(ihr, jhr, khr, nhr : byte);

begin
  if (nhr = 1)
    then begin
          writeln(ihr:2, ' -->', jhr:2); { ket korong eseten az atrakas jelzese }
          lepes := succ(lepes)
        end
    else begin
          hanoi_rec(ihr, khr, jhr, nhr-1);  { rekurziv eljaras elso lepese  }
          hanoi_rec(ihr, jhr, khr, 1);      { a maradek korong atrakasa     }
          hanoi_rec(khr, jhr, ihr, nhr-1)   { a rekurziv eljaras befejzese  }
        end
end;
{$a+}                              { csak CP/M gepen: rekurziv kod }

{ ============================================================ }

begin { main }

   clrscr;
   lepes := 0;

   write('Korongok szama = ');
   readln(n);
   writeln;

   hanoi_rec(1,2,3,n);
   writeln;
   writeln(lepes,' lepes');

   writeln;
   writeln('Nyomjon meg egy billenntyut!');
   repeat
   until keypressed;
   read(Kbd,ch)
end. { main }

{ ============================================================ }
```

```
program stack_proba;

type
  stack  = ^leiras;
  leiras = record
             adat   : integer;
             mutato : stack
           end;

var
   S1       : stack;
   im,mely : integer;
   ch       : char;
{ ========================================== }

procedure create(var S : stack);
begin
   new(S);
   S := nil
end;

{ ========================================== }

function Pop(var Spop : stack; npo : integer): integer;
var
   temp: stack;
begin
   if (Spop = nil)
     then Pop := npo
     else begin
            new(temp);
            Pop  := Spop^.adat;
            temp := Spop^.mutato;
            Spop := temp
          end
end;

{ ========================================== }

procedure Push(var Spush : stack; var npu : integer);
var
   temp : stack;

begin
   new(temp);
   temp^.adat   := npu;
   temp^.mutato := Spush;
   Spush        := temp
end;

{ ========================================== }
```

```
function IsEmpty(Se : stack) : Boolean;
begin
   if (Se = nil) then IsEmpty := true
                 else IsEmpty := false
end;

{ ======================================== }

begin {main program}
  create(S1);

  writeln;
  write('a stack melysege = ');
  readln(mely);
  writeln;

  for im := 1 to mely do Push(S1,im);

  while not IsEmpty(S1) do writeln(Pop(S1,0));

  writeln;

   writeln;
   writeln('Nyomjon meg egy billenntyut!');
   repeat
   until keypressed;
   read(Kbd,ch)
end.

{ ======================================== }
```

program CPM80Dir;

```
const
   Search_First   = $11;
   Search_Next    = $12;
   Set_DMA        = $1A;
   ScrSize        =  22;
   FnameSeparator = '.';
   FCB1           = $005C;

var
   Error,
   Dir_Entry : integer;
   FCB       : array[0..25]  of byte absolute FCB1;
   DMA       : array[0..$FF] of byte absolute $FE00;
   ch        : char;
{ ========================================= }

procedure Display_Fname(DMA_Shift : integer);

var Loop, Start : integer;
```

```
begin
  Start := DMA_Shift * $20;
  for Loop := Start to (Start + 8) do write(char(mem[addr(DMA)+Loop]));
  write(FnameSeparator);
  for Loop := (Start + 9) to (Start + 11) do write(char(mem[addr(DMA)+Loop]));
  writeln
end;

{ ========================================= }

procedure Page;

const CR = #$0D;

begin
  if (Dir_Entry = ScrSize)
    then begin
           Dir_Entry := 1;
           write(' -- more --');
           repeat
           until keypressed;
           write(CR,'                ',CR)
         end
    else Dir_Entry := succ(Dir_Entry)
end;

{ ========================================= }

begin
  Dir_Entry := 1;
  writeln;

  BDos(Set_DMA,Addr(DMA));

  FCB[0] := 0;
  FillChar(FCB[1],8 + 3,'?');
  Error := BDos(Search_First,Addr(FCB));

  if (Error = $FF) then Writeln(' No files')
                   else while (Error <> $FF)
                          do begin
                               Display_Fname(Error);
                               Page;
                               Error := BDos(Search_Next);
                             end;

  writeln;
  writeln('Nyomjon meg egy billenntyut!');
  repeat
  until keypressed;
  read(Kbd,ch)

end.
```

```
program graphic;

const
   chgsize  = $0800;
   chgstart = $3000;

type
   mode_type = (set_point, reset_point, change_point);
   filename  = string[12];
var
   ch        : char;
   i,j       : integer;
   chgsave  : array [1..chgsize] of byte;

{ ============================================ }

{$I install.inc }
{$I graphic.inc }

{ ============================================ }

begin { graphic }

   clrscr;
   install_code('graphic.hex');
   graphic(on);

   for i:=0 to 10 do GR_line(0,0,60*i,199);
   for i:=0 to 9 do GR_line(0,0,599,20*i);

   graphic(off);

end. {graphic}

{ ============================================ }
```

Az '**install.inc**' file az alábbi programot tartalmazza:

```
procedure install_code(code_file_name: filename);

type
   MaxString=      string[255];

var
   line:           MaxString;
   code_file:      text;
   i,j:            integer;
   memory:         integer;

function convert(var s: MaxString; i: integer): integer;
```

```
function hexval(ch: char): integer;

var
   temp: integer;

begin
   case ch of
     '0'..'9':    temp:=48;
     'a'..'f':    temp:=87;
     'A'..'F':    temp:=55;
    end;
    hexval:=byte(ch)-temp
end;

begin
   convert:=hexval(char(copy(s,i,1)))*16+hexval(char(copy(s,i+1,1)))
end;

begin
   assign(code_file,code_file_name);
   reset(code_file);

   while not EOF(code_file) do
      begin
         readln(code_file,line);
         i:=convert(line,2);
         if i>0 then
          begin
            memory:=convert(line,4)*256+convert(line,6);
            for j:=0 to i-1 do
               begin
                  mem[memory]:=convert(line,2*j+10);
                  memory := memory+1;
               end
          end
      end;

end;
```

A '**graphic.inc**' file az alábbi programot tartalmazza:

```
{ Low level routines to use the VDC memory for graphics }

procedure MoveTo(  video_to,
                   number,
                   memory_from : integer); external $E000;

procedure MoveFrom(video_from,
                   number,
                   memory_to   : integer); external $E003;


procedure VDC_reg_write(register,
```

```
                    data    : byte);        external $E006;

function  VDC_reg_read(register : byte) : byte; external $E009;


procedure GR_set_point(  pointer : integer;
                    bit     : byte);            external $E00C;

procedure GR_reset_point(pointer : integer;
                    bit     : byte);            external $E00F;

function  GR_test_point( pointer : integer;
                    bit     : byte) : boolean; external $E012;

{ ============================================ }

procedure ClrScrGraph;

var
   spaces: string[128];
   cicle: integer;

begin
   fillchar(spaces[1],128,0);
   spaces[0]:=char(128);
   for cicle:=0 to 128 do
      MoveTo(128 * cicle, 128, addr(spaces[1]))
end;

{ ============================================ }

procedure GR_point(x,y : integer; mode : mode_type);

var
   pointer:    integer;
   bit:        byte;

begin { GR_point }

   pointer := 80 * y + (x div 8);
   bit     := 7 - (x mod 8);
   case mode of
     set_point     : GR_set_point(pointer,bit);
     reset_point   : GR_reset_point(pointer,bit);
     change_point : if GR_test_point(pointer,bit)
                    then GR_reset_point(pointer,bit)
                    else GR_set_point(pointer,bit)
   end {case}

end;  { GR_point }

{ ============================================ }

const on  = TRUE;
```

```
      off = FALSE;

procedure graphic(switch : boolean);

const FG_BG   = 26;  { Foreground / Background color reg. }
      HOR_SCR = 25;  { Horizontal Smooth Scrolling reg.   }
      ATR_on  = $40; { Attribute Memory On                }
      ATR_off = $00; { Attribute Memory Off               }
      TXT_on  = $00; { Text Mode On                       }
      TXT_off = $80; { Text Mode Off                      }
      SMS_on  = $07; { Smooth Scrolling On                }
      WHITE   = $0F;
      BLACK   = $00;

begin { graphic }

   VDC_reg_write(FG_BG,(WHITE shl 4) + BLACK);
   case switch of
     on  : begin
             MoveFrom(chgstart,chgsize,addr(chgsave));
             VDC_reg_write(HOR_SCR,TXT_off + ATR_off + SMS_on);
             ClrScrGraph
           end;
     off : begin
             MoveTo(chgstart,chgsize,addr(chgsave));
             VDC_reg_write(HOR_SCR,TXT_on  + ATR_on  + SMS_on);
             ClrScr
           end
   end; { case }

end;  { graphic }

{ ========================================== }

procedure GR_line(x1,y1,x2,y2: integer);

   var x,y,z,a,b,dx,dy,d,
       deltap,deltaq : integer;

begin { line }
   dx:=abs(x2-x1);
   dy:=abs(y2-y1);
   if dy<=dx then
   begin
      x:=x1;
      y:=y1;
      z:=x2;
      if x1<=x2
         then a:=1
         else a:=-1;
      if y1<=y2
         then b:=1
         else b:=-1;
      deltap:=dy+dy;
```

```
      d:=deltap-dx;
      deltaq:=d-dx;
      GR_point(x,y,set_point);
      while x<>z do
      begin
         x:=x+a;
         if d<0
            then d:=d+deltap
            else
            begin
               y:=y+b;
               d:=d+deltaq
            end;
         GR_point(x,y,set_point);
      end { while }
   end { then }

   else
   begin
      y:=y1;
      x:=x1;
      z:=y2;
      if y1<=y2
         then a:=1
         else a:=-1;
      if x1<=x2
         then b:=1
         else b:=-1;
      deltap:=dx+dx;
      d:=deltap-dy;
      deltaq:=d-dy;
      GR_point(x,y,set_point);
      while y<>z do
      begin
         y:=y+a;
         if d<0
            then d:=d+deltap
            else
            begin
               x:=x+b;
               d:=d+deltaq
            end; { else }
         GR_point(x,y,set_point);
      end { while }
   end { else }
end; { line }

procedure GR_ellipse(x0,y0,rx,ry: integer; alfa: real);

const pi=3.141592;
      di=3;

var x,y,xe,ye,xk,yk,xke,yke,h,dfi,
    SinDfi,CosDfi,SinAlfa,CosAlfa: real;
```

```
    i,n: integer;

begin
   h:=ry/rx;
   dfi:=di/rx;
   n:=2*trunc(abs(pi/dfi)+0.5);
   dfi:=pi/n*2;
   SinDfi:=sin(dfi);
   CosDfi:=cos(dfi);
   SinAlfa:=sin(alfa);
   CosAlfa:=cos(alfa);
   xke:=rx;
   yke:=0;
   xe:=h*xke*CosAlfa-yke*SinAlfa;
   ye:=yke*CosAlfa+H*xke*SinAlfa;

   for i:=1 to n do
   begin
      xk:=xke*CosDfi-yke*SinDfi;
      yk:=yke*CosDfi+xke*SinDfi;
      x:=h*xk*CosAlfa+h*xk*SinAlfa;
      y:=yk*CosAlfa+h*xk*SinAlfa;
      GR_line(round(xe+x0),round(ye+y0),round(x+x0),round(y+y0));
      xke:=xk;
      yke:=yk;
      xe:=x;
      ye:=y
   end
end; {ellipse}
```

15. fejezet

Makro assembler fejlesztő rendszer

A CP/M rendszerhez külön megvásárolható 'Additional Utilities' lemez tartalmazza a makro assembler, továbbá a CP/M új változatának generálásához szükséges programokat. A fejlesztő rendszer további két - önmagában semmire nem használható - programja az eredeti lemezen található.

Sorban ismertetjük az egyes programok használatát.

GENCOM

A program segítségével .COM típusú file-hoz memória rezidens bővítést, .RSX-t adhatunk hozzá. A töltő (vagyis a CCP.COM program) érzékeli az .RSX bővítést és a megfelelő helyre tölti azt.

Szintaxis:

GENCOM <file> <bővítés> {**[LOADER|NULL|SCB=(**<mut>,<érték>**)]**}

A <file> a módosítandó parancs file neve, míg a bővítés .RSX file-ok sorozata. Egy paranccsal legfeljebb 15 .RSX bővítést tudunk a file-hoz hozzátenni.

A LOADER paraméter használatakor a töltes-mutató aktív marad. A NULL használata jelzi, hogy csak .RSX file-ok vannak a parancsban. Ilyenkor a <file>-nak is egy .RSX bővítésnek kell lennie.

A parancs hatására az első file nevével azonos, de .COM kiterjesztésű file-t hoz létre a GENCOM, amely file-hoz már hozzáadta a szükséges rezidens bővítéseket. Ha a <file> maga is .COM típusú, akkor az eredeti file elvész.

Az SCB paraméterrel a rendszer kontroll blokkjának adott paraméterét módosíthatjuk.

HEXCOM

A parancs a paraméterként megadott szabvány Intel .HEX formájú file-t transzformálja át .COM típusú file-ja. A file kiterjesztését nem kell megadni, mert az csak .HEX lehet.

Szintaxis:

HEXCOM <file>

LIB

A parancs segítségével MicroSoft relokálható formában (REL) levő könyvtárakat hozhatunk létre, illetve az abban levő modulokat kezelhetjük.

Szintaxis:

LIB <file> {[I|M|P|D]}
LIB <file> {[I|M|P]}=<file1>{<mód>}{,<file2>{<mód>...}

A könyvtári file-ok vagy .REL vagy .IRL kiterjesztésűek. A .REL típusú könyvtárban levő hivatkozások nem mutathatnak visszafelé, mert a LINK program csak egyszer olvassa végig a könyvtárat.

A parancsban az egyes paraméterek jelentése az alábbi:

I =INDEX. Hatására a létrehozandó új könyvtári file **indexelt** lesz (és így .IRL) típusú. Az indexelt könyvtárak nagyobbak, de gyorsabb bennük a keresés.
M =MODULE. A paraméter használatakor a rendszer kijelzi a modulok nevét.
P =PUBLICS. Hatására a könyvtári modulok nevét és a megtalált PUBLICS típusú változókat írja ki a rendszer.
D =DUMP. A paraméter használatakor a program ASCII formában kiírja a modulok tartalmát.

A parancs második formában való használatakor <file> nevű új könyvtárat hozunk létre a már meglevő <file1>, <file2> stb. könyvtárakból. A file-nevek típusát .REL-nek tételezi a program, megadásuk csak akkor szükséges, ha ettől eltér.

A <mód> paraméter adja meg, hogy milyen műveletet kell az adott file-lal végezni.

nincs <mód> megadva A file valamennyi modulja bekerül az új könyvtárba.

<név=> A név nevű modul kivételével valamennyi modult átmásolja a rendszer a file-ból az új könyvtári file-ba. A < és > jelek a parancs részei!
<név=file.REL> A file moduljait a 'név' nevű modul kivételével átmásolja a program. A 'név' modul helyett pedig a 'file' .REL típusú file kerül a könyvtárba. A < és > jelek a parancs részei!
(item1,item2,...) Az item1, item2 stb. adatok vagy modulnevek, vagy modul1-modul2 alakúak. A file-ból csak a felsorolt modulok kerülnek át az új könyvtárba.

LINK

A program segítségével a MicroSoft REL formátumú file-okból futtatható – .COM típusú – file-t állíthatunk elő.

Szintaxis:

LINK {<file>,{[<opció>]}=}<file1>{[<opció>]}{,...}

A parancs fenti alakjában a <file1>, <file2>,... file-okból állít elő a program egy futtatható, .COM típusú file-t a parancsban használt <opció>-k szerint. Az új file neve <file> lesz. Ha a <file>-t nem adjuk meg, akkor a futtatható file neve <file1>.COM lesz.

Az egyes <opció>-k a következők lehetnek:

A
A program nem használ puffert, valamennyi ideiglenes adatot a lemezre ír.

B

BIOS összefűzése. SPR file-ok előállítására szolgál.

Dhhhh

A DATA és a COMMON szegmens kezdete hexadecimális alakban.

Gn

A program indítási pontjának a meghatározása. n a programban szereplő címke.

Lhhhh

A modul töltési címét 100H-ról 0hhhhH-ra változtatja.

Mhhhh

Memória méret MP/M modulok számára.

NL

A szimbolum táblát a rendszer nem listázza ki a konzolra.

NR

Nem hoz létre szimbolum tábla file-t.

OC

A szerkesztő COM típusú file-t hoz létre.

OP

A szerkesztő laphatárra relokálható file-t hoz létre (PRL).

OR

A szerkesztő rezidens rendszer-processz file-t (RSP) hoz létre.

OS

A szerkesztő rendszer laphatárra relokálható (SPR) file-t hoz létre.

Phhhh

Program kezdet megadása. A program kezdetét 100H-ról 0hhhhH-ra változtatja.

Q

Kilistázza a ?-jel kezdődő szimbólumokat.

S

A megelőző file-t könyvtárként kezeli, s csak a hiányzó modulokat válogatja ki belőle.

$Cd

A konzol üzenetek helye. d=X a konzol, d=Y a nyomtató, d=Z nincs. Alapértelmezése X.

$Id

A közbenső file-ok lemezegysége.

$Ld

A könyvtári file-okat tartalmazó lemezegység.

$Od

A létrehozott új file lemezegysége. Alapértelmezése a <file1> által meghatározott lemezegység.

$Sd

A szimbolum táblát tartalmazó file lemezegysége.

MAC

A MAC program a CP/M makro assemblere.

Szintaxis: **MAC** <file> {**$**<opciók>}

A MAC kizárólag .ASM típusú file-t fordít s HEX típusú file-ra. A lefordított file-neve <file>.HEX lesz. A fordítási listát <file>.PRN nevű file-ba helyezi el, míg a szimbolum táblát tartalmazó file neve <file>.SYM. Az <opciók> megadásával ezeknek a file-oknak a helyét adhatjuk meg. Az opciókat vesszővel kell elválasztani egymástól. A használható opciók a következők:

Ad

A forrás file (.ASM) helye. d=A-O.

Hd

AA fordítás (.HEX) helye. (d=A-O vagy Z. Z esetén nem hoz létre file-t.

Ld

A makro könyvtárakat tartalmazó file-ok helye. d=A-O.

Pd

A forráslista (.PRN) helye. d=A-O,X,P,Z. X=konzol, P=nyomtató, Z=nincs.

Sd

A szimbolum táblát tartalmazó file helye. d=A-O,X,P,Z. X=konzol, P=nyomtató. Z=nincs.

+L

A makro könyvtárból olvasott sorokat kilistázza.

-L

A makro könyvtárból olvasott sorokat nem listázza ki.

+M

Az összes makro sort kilistázza, ahogy a fordítás során megváltozott.

-M

A fenti listázásra nem kerül sor.

***M**

A makrok által generált hexadecimális értékeket listázza csak ki.

+Q

Valamennyi LOCAL szimbolumot is kilistázza.

-Q

A LOCAL szimbolumokat nem listázza ki.

+S

A szimbolum táblát a forráslista végére fűzi.

-S

Nem hozza létre a szimbol tábla file-ját.

+1

A PRN file-ban az első menetet is kilistázza a makrok nyomkövetéséhez.

-1

A PRN file-ban az első menet listája nem íródik ki. (Alapértelmezés).

RMAC

A CP/M relokálható makro assemblere. Segítségével .ASM típusú file-okat fordít .REL típusú MicroSoft relokálható formátumú file-ba.

Szintaxis: **RMAC** <file> {**$** **R**d| **S**d| **P**d}

Az egyes paraméterek jelentése a következő:

Rd

A fordított, REL típusú file helye. d=A-O,Z. Z=nincsen file generálás, csak szintaktikus ellenőrzés.

Sd

A szimbolum táblát tartalmazó file helye. d=A-O,X,P,Z. X=konzol, P=nyomtató, Z=nincs.

Pd

A fordítási listát tartalmazó file helye. d=A-O,X,P,Z. X=konzol, P=nyomtató, Z=nincs.

SID

A SID (**S**ymbolic **I**nstruction **D**ebugger) a CP/M monitora, s lehetővé teszi a MAC vagy RMAC által előállított szimbolum táblák használatát. Nagy hiányossága, hogy nem képes a Z80 kódokat visszafordítani. Ezért helyette a hasonlóan működő ZSID monitort szokás használni. Egyetlen lényeges eltérés a kettő közt, hogy a ZSID Z80, míg a SID Intel mnemonikokkal dolgozik!

Szintaxis: **SID** <program>{,<szimbol>}

Ha nem adjuk meg a <program> és a <szimbol> paramétereket, akkor a SID a memóriába tesztprogram nélkül töltődik be. A <program> .COM vagy .HEX típusú file lehet. A <szimbol> a szimbolum táblát tartalmazó file neve, a kiterjesztését nem kell megadni, a SID mindig .SYM-nek tekinti.

A SID futása közben a <CTRL-S> megállítja, míg a <CTRL-Q> újraindítja a képernyőre való kiírást. (Használható erre a <NoScroll> billentyű is!) A SID-ből a <CTRL-C> megnyomásával térhetünk vissza a CP/M-be.

A monitor parancsok paraméterei számok, a szimbolum táblában tárolt nevek és ezek közti műveletek lehetnek. Ha egy szimbólikus nevet a @ jellel előzünk meg, akkor az a név által megjelölt és az azt követő memóriahelyen tárolt 16 bites értéket jelenti. Ha a nevet egy = jellel előzzük meg, akkor a névnek megfelelő memóriahelyen levő 8 bites értékkel tér vissza.

A SID hexadecimális számokkal dolgozik. Ha a szám, amit használunk decimális, akkor # jellel kell bevezetni. Pl. 0ABC vagy #121.

A számok, nevek értékét összeadhatjuk (+), kivonhatjuk egymásból (–). Egy speciális jelet, a felfelé mutató nyilat (↑) használjuk a verem tetején levő elem értékére. A kifejezésekben használt ↑ nem módosítja a verem tartalmát. Pl. ↑↑↑ a verem harmadik elemének értékét veszi fel.

A továbbiakban felsoroljuk a SID utasításait:

As

Az utasítás kiadása után lehetőségünk van assembly utasítások beírására az s címtől kezdve. A beírást egy üres sor bevitelével fejezhetjük be.

Cs{b{,d}}

Szubrutinhívás az s címen levő utasításra. b a BC, d a DE regiszterekbe betöltendő értéket jelenti.

D{W}{s}{,f}

Memória tartalmának kijelzése. s a kezdő, f a végcím. Ha a W-t is megadjuk, akkor 16 bites címeket jelez ki a monitor.

E<program>{,<szimbolum>}

Program és szimbolum tábla betöltése a memóriába.

E*<szimbolum>

Csak szimbolum táblát tartalmazó file betöltése a memóriába.

Fs,f,d

Adott memóriarész feltöltése a d karakterrel (8 bites értékkel) s a kezdeti, f a végcím.

G{p}{,a{,b}}

Program indítása a p címtől. Ha a p-t nem adjuk meg, akkor az utasításszámlálónak a monitor által tárolt értékétől fog elindulni. a és b két ideiglenes töréspont címe. A program az elsőnek elért ideiglenes vagy állandó töréspontig fut.

Ha
Ha,b

Az első esetben a monitor kijelzi az a értékét hexadecimális, decimális és ASCII alakban, míg a második esetben kijelzi a és b összegének és különbségének hexadecimális alakját.

I<parancs>

A megadott <parancs> lesz a CCP bufferében levő karaktersorozat. Segítségével szimulálni lehet paramétereket is feldolgozó programok működését.

L{s},{f}

Memória vissza-assemblálása. s a kezdőcím, f a végcím. Ha nincs megadva az s, akkor a listázás a monitor által tárolt utasításszámláló értékétől indul.

Ms,h,d

Memóriatömb mozgatása. s és h a mozgatandó memória első és utolsó byte-ja, míg d az új memóriarész első címe.

P{p{,c}}

Állandó töréspont definiálása. p a töréspont sorszáma, c az ismétlési tényező.

R<file>{,d}

Program/szimbolum olvasása. A d valamennyi cím értékét módosítja.

S{**W**}{s}

A memória adott részébe adatok beírása. s a kezdőcím. Ha W-t megadjuk, akkor 16 bites címeket írhatunk be.

T{n{,c}}

Program nyomkövetése. n a követendő programlépések száma, c a utility program belépési pontja. A nyomon követett program minden egyes lépésének végrehajtása után ez az alprogram hívódik meg.

TW{n{,c}}

Program nyomkövetése. n a követendő programlépések száma, c a utility program belépési pontja. A program minden egyes lépésének végrehajtása után ez az alprogram hívódik meg. A W hatására a szubrutinhívásokat nem követi a rendszer.

U{**W**}{n{,c}}

Program nyomkövetése a végrehajtás kijelzése nélkül. Hatása különben megegyezik a T és TW monitor parancsokkal.

V

Kijelzi a következő elérhető memória címet (NEXT) az olvasott file-ok után elérhető első címet (MSZE), illetve az egyáltalán használható memória (END) címét.

W<file>{,s,f}

A <file>-ba kiírja a memória adott részét. Az első és utolsó kiírt memória címe s illetve f.

X{f}{r}

A processzor regisztereinek/mutatóinak lekérdezése/beállítása. f=C,E,I,M, vagy Z, míg r=A,B,D,H,P vagy S lehet.

XREF

Az utasítás az RMAC vagy a MAC által előállított .PRN és .SYM file-okból keresztivatkozási táblázatot készít.

Szintaxis: **XREF** {d:}<file>{**$P**}

A SYM és PRN file-nak ugyanolyan file-nevűnek kell lennie, mint a <file> file-név. Ha a $P opciót megadjuk, akkor a kereszthivatkozási táblázat a nyomtatóra kerül.

Kiadó: LSI ATSz
Felelős kiadó: Dr. Kovács Magda
Lektor: Baki Zoltán
Témafelelős és
technikai szerkesztő: Nagy Olivér
Borítóterv: Székely Edith
ISBN: 963 592 6898
Készült a Szabadsag MGTSZ Nyomdaüzemében, Gyál 88-9/sz.
Felelős vezető: Tóth Antal

Ára: 185 Ft


COMPUTER
COMPUTER
COMPUTER
MIKROSZÁMÍTÓGÉPEK
PERIFÉRIÁK
BŐVÍTÉSEK
FLOPPY DISZK
KAZETTA
FESTÉKSZALAGOK
PROGRAMOK
SZAKKÖNYVEK
TANÁCSADÁS
NAGYMEZŐ UTCA
64
VÉTEL – ELADÁS